TURING 图灵程序设计丛书

[PACKT] PUBLISHING

[印尼] Ivan Idris 著　张崇明 译

NumPy Cookbook

NumPy攻略

Python科学计算与数据分析



人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

Ivan Idris

实验物理学硕士。先后任职于多家公司，从事Java开发、数据仓库开发和QA分析等工作。主要的兴趣是商业智能、大数据和云计算，喜欢编写整洁、可测试的代码，以及撰写有趣的技术文章。另著有*NumPy Beginner's Guide*和*Instant Pygame for Python Game Development How-to*等书。可以访问ivanidris.net获取更多信息。

张崇明

本科及研究生毕业于天津大学精密仪器与光电子工程学院，博士毕业于复旦大学计算机科学技术学院。在中兴通讯南京研发中心做过三年通信软件的开发。目前在上海师范大学信息与机电工程学院从事教学和科研工作。

TURING 图灵程序设计丛书

[印尼] Ivan Idris 著　张崇明 译

NumPy Cookbook

NumPy攻略

Python科学计算与数据分析

人民邮电出版社
北京

图书在版编目（ＣＩＰ）数据

NumPy攻略 : Python科学计算与数据分析 / (印尼)
伊德里斯（Idris,I.）著 ; 张崇明译. -- 北京 : 人民
邮电出版社，2013.10
（图灵程序设计丛书）
书名原文: NumPy Cookbook
ISBN 978-7-115-32991-2

Ⅰ. ①N… Ⅱ. ①伊… ②张… Ⅲ. ①软件工具－程序
设计 Ⅳ. ①TP311.56

中国版本图书馆CIP数据核字(2013)第211206号

内 容 提 要

本书介绍了70多种学习Python开源教学库NumPy的有趣方法，教会读者如何安装和使用NumPy，并了解其他一些相关概念，进而掌握NumPy arrays及其通用功能，书中的例子还涉及Matplotlib、SciPy等Python科学计算生态系统中的其他重要软件。此外，还介绍了NumPy和其他软件的交互、性能分析和调试、软件测试和Cython等比较高阶的话题。

本书的目标读者是对Python和NumPy有基本了解，并且希望自己的水平能更上一层楼的科技工作者、工程师、程序员和分析师。

◆ 著　　　　[印尼] Ivan Idris
译　　　　张崇明
责任编辑　李　瑛
责任印制　焦志炜

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街14号
邮编　100061　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　　　印刷

◆ 开本：800×1000　1/16
印张：11.75
字数：277千字　　　　2013年10月第1版
印数：1－3 000册　　　2013年10月北京第1次印刷

著作权合同登记号　图字：01-2013-4681号

定价：49.00元

读者服务热线：(010)51095186转604　印装质量热线：(010)67129223

反盗版热线：(010)67171154

广告经营许可证：京崇工商广字第0021号

版 权 声 明

译　者　序

这是我的第一本译著。本着先易后难的原则，我从图灵社区开放出版的书目中，选择了*NumPy Cookbook*这本书来翻译。虽然我对NumPy不太熟悉，但想来这样一本200多页、介绍单一小众软件的书，翻译起来应该没多大难度。很遗憾，随着翻译工作的进行，我发现自己最初的想法完全不对。

首先，用小众软件形容NumPy并不妥当。作为Python科学计算生态系统的重要组成部分，NumPy具备强大的多维数组功能，提供了十分丰富的对数组进行处理和操作的函数集。可以说，用Python做科学计算时，到处可以看到NumPy的身影（看过图灵出版的《机器学习实战》一书的读者应该会同意这个说法）。因此，如果你想使用Python做科学计算相关的工作，NumPy是必须要掌握的。

其次，本书并不只介绍NumPy这一个软件。Python科学计算生态系统中的其他重要软件，如数值计算库SciPy、符号计算库SymPy、绘图库Matplotlib和各种scikit项目（机器学习、统计建模、图像处理、数据分析）等，也都有不同程度的介绍。此外，还介绍了NumPy和其他软件的交互、性能分析和调试、软件测试和Cython等比较高阶的话题。总之，本书涉及的话题相当广泛，这给翻译工作带来了很大的麻烦。我除了要翻看1400多页的NumPy官方参考文档之外，还要浏览书中介绍的各种软件的参考文档和源代码。希望这对读者是好事，一下可以了解Python科学计算生态系统中这么多内容。

虽然本书介绍了如此多的内容，但篇幅却不长。本书的大多数攻略都是一个完整的、可以运行的程序，作者对各种知识点的介绍都很简洁，有些地方只是点到为止，甚至有些地方只是给出一个维基百科的网页链接让你自己去看。希望读者不要因为有些细节没搞明白而感到沮丧，而是把这本书看作一个了解Python科学计算生态系统的快速通道。在此基础上，对于感兴趣的内容，自己再进一步深入学习。例如，看过《机器学习实战》的读者，很可能对本书介绍的scikit-learn项目感兴趣，那就去scikit-learn.org看看官方的参考文档，然后去Github把源代码下载下来深入研究吧。

本书用了一定的篇幅介绍各种软件在各种操作系统中的安装过程。我认为读者不必这样做，推荐大家直接安装一个Python的发行版本，例如EPD（Canopy）、Python(x,y)或Anaconda。本书需要用到的大多数软件这些发行版中都有，然后再参考相关的攻略，缺什么安装什么，这样比较有

效率。我选用的是Anaconda，使用效果良好。

虽然翻译过程没有想象的顺利，但我本人还是因此收获颇丰。收获之一是对Python生态系统更加了解了。Python算是比较老的脚本编程语言了，长期以来，我对Python的使用也仅限于快速做一些简单的数据处理工作。通过这两个多月对相关资源的广泛涉猎，我感觉到了这个不算年轻的编程语言的迅猛发展势头。我本人非常看好Python开源科学计算生态系统的发展潜力。另外一个收获是，我发现本书居然和大数据、云计算和物联网等时髦概念都有点关系，满足了我学习了解新事物的嗜好。本书用比较多的篇幅介绍了数据分析相关的内容，也在介绍谷歌应用程序引擎和PiCloud等内容时涉及了云计算的话题，但看似和物联网没关系。而实际上，Python在物联网方面的应用实例已经比较多了。物联网终端设备会产生大量的数据，要求上位机软件有强大的数据处理能力。Python作为一个“胶水语言”，既可以方便地与各种物联网终端的硬件接口程序打交道，也可以方便地使用“NumPy及它的小伙伴们”在本地或在云端对数据进行处理，是做物联网应用的理想编程语言。

现在相关的图书资料还相当缺乏。本书作者显然想在第一时间和大家分享他的知识，因此本书的有些地方有一定的改进空间，甚至还有一些错误。在翻译过程中，我已经修改了多处比较明显的小错误，希望中文版看上去并不比英文版难懂。因为译者水平精力都有限，可能会在翻译过程中引入新的错误，恳请大家在阅读过程中发现问题后不吝赐教。既可以在图灵社区提交勘误，也可以用email向我反馈（czhang@shnu.edu.cn）。非常感谢！

顺便说一下，我觉得本书也适合作为普通高等院校任意选修课的教材。我准备面向本科生开设一门名为“Python数据分析”的选修课，计划安排36个学时，全部在机房上课。通过攻略的方式组织教学内容比较符合现在大学生的学习习惯，希望能有比较好的实施效果。

最后要感谢图灵公司武卫东、朱巍、岳新欣和姜新农等出版界同仁的帮助，感谢各位在翻译和审校等环节的辛勤付出。感谢上海师范大学科技处、教务处和信息与机电工程学院相关领导和同事的支持。感谢智翔集团张钦礼总经理提供的各种帮助和便利条件。感谢我的家人对我的支持。

张崇明

2013年7月于上海

前　　言

作为NumPy的使用者，我们正生活在一个令人兴奋的时代。每周甚至每天，似乎都有新的NumPy相关的开发进展引起我们的关注。就在本书写作期间，NumFOCUS基金会（NumPy Foundation of Open Code for Usable Science）成立了，基于LLVM框架并且支持NumPy的动态Python编译器项目Numba宣告启动，谷歌公司在其云计算产品Google App Engine中也增加了对NumPy的支持。

我们预计，NumPy将会改进对GPU和CPU集群的并发性支持，类似OLAP的查询操作也将有可能支持NumPy数组类型的使用。

这是一个好消息。不过我们也要意识到，在Python科学软件生态系统中，NumPy只是其中一员。除了NumPy，还有SciPy、Matplotlib（一个非常有用的Python绘图库）、IPython（一个交互式Shell）和Scikits等。在Python生态系统之外，诸如R、C和Fortran等编程语言也非常流行。我们将会讨论与这些编程环境交换数据的细节。

本书内容

第1章“使用IPython”介绍了IPython的使用。IPython是一个工具集合，因为它的Shell而为人所知。基于Web的notebook是个令人兴奋的新特性，我们将会对此做详细介绍。在Matlab和Mathematica软件中都有类似的notebook界面，但在IPython中我们是在浏览器里使用一个开源且免费的notebook。

第2章“高级索引和数组概念”介绍了NumPy中高级而巧妙的索引技术。由于使用了性能优异的索引技术，NumPy中的数组类型的实现非常高效，并且易于使用。

第3章“常用函数”对每一位NumPy使用者都应该知道的最基本的函数进行了介绍。NumPy中包含的函数太多了，不可能在本书中一一提及。

第4章“NumPy与其他软件的交互”。在实际工作中，我们需要用到各种编程语言、库文件和工具软件，数量多得惊人。一些软件运行在云端，另一些运行在本机或者远程服务器上。知道怎样在这样的软件环境中使用NumPy是十分重要的，其重要性不亚于能够编写独立运行的NumPy程序。

第5章“声音和图像处理”让你从一个不同的视角看待NumPy。看过本章内容后，当你想到NumPy时，很可能同时联想到声音和图像。

第6章“特殊类型数组与通用函数”探讨特殊类型数组和通用函数等话题。这将有助于我们学习怎样进行字符串操作、忽略不合法的数值和存储异构数据。

第7章“性能分析与调试”将介绍几个实用的性能分析和调试工具，它们是编写优秀的应用软件所必需的工具。

第8章“质量保证”将讨论单元测试、模拟和BDD等常用方法与技术，还会介绍NumPy中的测试工具，因为质量保证值得我们密切关注。

第9章“用Cython为代码提速”从NumPy的视角展示了Cython是怎样工作的。Cython试图把C语言的速度优势和Python的强大功能结合起来。

第10章“有趣的Scikits”对几个最有用的Scikits项目作了简明的介绍。Scikits同样属于令人着迷的Python科学计算生态系统。

本书需要的资源

为了运行本书的范例程序，你需要有新近版本的NumPy，这意味着你也需要有支持该版本NumPy的Python版本。其他需要用到的软件包的安装方法会在书中介绍。

本书读者

本书的目标读者是对Python和NumPy有基本了解，并且希望自己的水平能更上一层楼的科技工作者、工程师、程序员和分析师。此外，也需要读者对数学和统计学比较熟悉，或至少有点兴趣。

排版约定

本书针对不同种类的信息，采用了不同的排版样式。下面是一些不同排版样式的例子及对应的说明。

正文中的代码文字会这样显示：“我们可以使用`include`指令将其他内容包括进来。”

代码段会这样显示：

```
[default]
exten => s,1,Dial(Zap/1|30)
```

```
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

当我们想让你关注代码段的特定部分时，相关的行或条目会加粗显示：

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

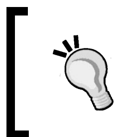
命令行输入或输出会这样显示：

```
# cp /usr/src/asterisk-addons/configs/cdr_mysql.conf.sample
    /etc/asterisk/cdr_mysql.conf
```

新术语用楷体字显示。重要的英文文字会加粗显示。例如，显示在屏幕上、菜单里或对话框中的文字在正文中出现时，会这样排版："点击Next按钮进入下一屏。"



警告或重要的说明在这里显示。



提示和小技巧在这里显示。

读者反馈

我们很欢迎读者的反馈。请告诉我们你觉得本书怎么样，你喜欢哪些内容，不喜欢哪些内容。读者的反馈对于我们策划出版对读者真正有用的图书至关重要。

如果向我们提供一般的反馈，给feedback@packtpub.com这个邮箱发邮件即可。请在邮件的标题中指明相应的书名。

如果你擅长某类选题并且有兴趣写书或者参与书籍的撰稿工作，请访问www.packtpub.com/authors，看看我们的作者指南。

客户支持

你现在是Packt出版社图书产品的尊敬用户，为使你的购买物超所值，我们会为你提供一些增值服务。

下载范例代码

访问http://www.packtpub.com并登录账号，你可以下载到所有已购图书中的范例代码。如果你是在其他地方购买的本书，可以访问http://www.packtpub.com/support并进行注册，相关的范例代码会直接通过邮件发给你。

勘误

尽管我们已经非常小心谨慎，尽力确保内容的准确性，但错误还是不可避免。如果你在书中发现了错误（可能是文字或者代码方面的错误），并向我们报告，我们将感激不尽。这样做，可以使其他读者免受这些错误的困扰，帮助改善该书的后续版本。如果你发现了任何错误，请访问http://www.packtpub.com/support，选择书名，点击“勘误提交表”链接，然后输入你所发现错误的详细内容。一旦你指出的错误被确认，你的提交就会被接受，勘误信息就会上传到我们的网站或添加到该书勘误区中已经存在的勘误列表中。

关于盗版行为

对于各种媒体，互联网上受版权保护的材料都长期面临非法复制的问题。Packt非常重视版权保护和版权许可。如果你在网上看到Packt图书的任何形式的非法复制，请立即向我们提供网址信息，以便我们及时补救。

请通过copyright@packtpub.com联系我们，并提供疑似盗版材料的链接信息。

我们感谢你的帮助。这将保护我们作者的利益，也使我们有能力为你提供有价值的内容。

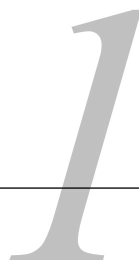
如果你有疑问

如果你对本书有任何疑问，可以通过questions@packtpub.com联系我们，我们会尽力为你解答。

目　　录

第 1 章

使用IPython



本章主要内容：

- 安装IPython
- 使用IPython的shell
- 阅读手册页
- 安装Matplotlib
- 运行基于Web的notebook
- 从notebook导出脚本和数据
- 导入脚本和数据到notebook
- 配置notebook服务器
- 初探SymPy配置

1.1 引言

IPython是一个免费、开源的项目，支持Linux、Unix、Mac OS X和Windows平台，其官方网址是http://ipython.org/。IPython的作者只要求你在用到IPython的科技著作中注明引用即可。IPython中包括各种组件，其中的两个主要组件是：

- 基于终端方式和基于Qt的交互式Python shell
- 支持多媒体和绘图功能的基于Web的notebook（版本号为0.12以上的IPython支持此功能）

与IPython兼容的Python版本是2.5[①]、2.6、2.7、3.1和3.2。

不需要本地安装，你可以在云端尝试使用IPython，网址为http://www.pythonanywhere.com/try-ipython/。和本地安装的IPython相比，云端版本会稍有时延，使用体验稍逊，但已具备IPython交互式shell的绝大多数功能。在云端版本中还可使用vi/vim编辑器。如果你喜欢vi，这自然是个很棒的功能，你可以在IPython会话过程中保存和编辑文件。只有vi编辑器可用，对我来讲不是什

① IPython的较新版本已不支持Python 2.5。——译者注

么问题，我本人对Emacs之类的其他编辑器并不感兴趣。

1.2 安装 IPython

IPython有许多种安装方式，这主要和使用什么操作系统有关。基于终端的shell组件依赖于readline的存在，基于Web的notebook需要用到tornado和zmq。

除了安装IPython，我们还需要安装setuptools，其中包含了easy_install命令。easy_install是Python默认的标准化的包管理器。easy_install安装好之后，继续安装pip。pip和easy_install命令的功能类似，但增加了一些选项，例如卸载。

1.2.1 具体步骤

本节将介绍在Windows、Mac OS X和Linux环境中怎样安装IPython，怎样使用easy_install和pip安装IPython及其依赖文件，以及怎样直接用源文件安装。

▶ 在Windows中安装IPython和setuptools

在IPython的官网可以下载适用于Python 2和Python 3的二进制Windows安装文件。具体安装过程请参阅http://ipython.org/ipython-doc/stable/install/install.html#windows。

从https://pypi.python.org/pypi/setuptools#files获得setuptools的安装文件并完成安装。之后继续安装pip，具体步骤为：

```
cd C:\Python27\scripts
python .\easy_install-27-script.py pip
```

▶ 在Mac OS X中安装IPython

如有必要，请先安装苹果开发工具Xcode，可以在Mac电脑附带的OSX DVD光盘中或者苹果应用商店中找到Xcode。按照本节后面的说明，使用easy_install或pip安装IPython，或者从源文件安装。

▶ 在Linux中安装IPython

Linux的发行版本众多，恕不能一一列举。

- Debian版本的安装命令如下：

  ```
  su - aptitude install ipython python-setuptools
  ```

- Fedora版本的安装命令如下：

  ```
  su - yum install ipython python-setuptools-devel
  ```

- Gentoo版本的安装命令如下：

```
su - emerge ipython
```

- Ubuntu版本的安装命令如下：

```
sudo apt-get install ipython python-setuptools
```

▶ 使用easy_install或pip安装IPython

使用easy_install安装IPython和本章中各种攻略所需要的依赖文件，使用以下命令：

```
easy_install ipython pyzmq tornado readline
```

或者你可以先用easy_install安装pip，在终端界面中键入以下命令：

```
easy_install pip
```

之后使用pip安装IPython，命令如下：

```
sudo pip install ipython pyzmq tornado readline
```

▶ 从源文件安装

如果你想使用最新的开发版本，从源文件安装是最适合的。

1. 从https://github.com/ipython/ipython/downloads下载最新的压缩包。
2. 对下载的文件解压缩，获得源文件：

```
tar xzf ipython-<version>.tar.gz
```

3. 如果你已经安装了Git，也可以通过克隆Git版本仓库的方式获得源文件：

```
$ git clone https://github.com/ipython/ipython.git
```

4. 进入ipython目录：

```
cd ipython
```

5. 运行安装脚本。你可能需要使用sudo运行此脚本，命令如下：

```
sudo setup.py install
```

1.2.2 攻略小结

本节介绍了安装IPython的各种方法。大多数方法安装的是最新的稳定版本。如果选择从源文件安装，你安装的就是最新的开发版本。

1.3　使用 IPython 的 shell

科学家和工程师习惯于做各种实验。正是一些总有实验想法的科学家编写了IPython。IPython提供的交互式实验环境，使其具备了与Matlab、Mathematica、Maple和R类似的使用体验。

IPython的shell具有以下特性。

- 代码补全
- 历史记录机制
- 嵌入式编辑
- 使用`%run`调用外部Python脚本的能力
- 访问系统命令
- pylab选项开关
- 访问Python的调试器和性能分析器

1.3.1　具体步骤

本节具体介绍怎样使用IPython的shell。

▶ pylab选项开关

使用pylab选项开关可以自动引入SciPy、NumPy和Matplotlib软件包。如果不使用这个选项开关，就需要自己引入这些软件包。

我们只需要在命令行输入以下指令：

```
$ ipython -pylab
Type "copyright", "credits" or "license" for more information.

IPython 0.12 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

Welcome to pylab, a matplotlib-based Python environment [backend: MacOSX].
For more information, type 'help(pylab)'.
In [1]: quit()
quit() or Ctrl + D quits the IPython shell.
```

▶ 保存会话

我们也许需要回溯做过的实验。要在IPython中保存会话以供将来使用，只需输入以下命令：

```
In [1]: %logstart
Activating auto-logging. Current session state plus future input saved.
Filename       : ipython_log.py
Mode           : rotate
Output logging : False
Raw input log  : False
Timestamping   : False
State          : active
```

关闭日志记录的命令如下：

```
In [9]: %logoff
Switching logging OFF
```

▶ 执行系统shell命令

在使用默认设置的IPython环境中执行系统shell命令时，要在系统命令前加!前缀。例如，以下输入将获得当前日期：

```
In [1]: !date
```

实际上，以!为前缀的任何内容都被发送到了系统shell。命令执行后的输出结果也可以进行保存，如下所示：

```
In [2]: thedate = !date
In [3]: thedate
```

▶ 显示历史记录

使用%hist命令，可以显示命令的历史记录，例如：

```
In [1]: a = 2 + 2

In [2]: a
Out[2]: 4

In [3]: %hist
a = 2 + 2
a
%hist
```

这是命令行接口（CLI）环境的一个常见特性。使用-g选项可以实现对历史记录的搜索：

```
In [5]: %hist -g a = 2
    1: a = 2 + 2
```



下载范例代码

访问http://www.packtpub.com并登录账号，可以下载到所有已购图书中的范例代码。如果你是在其他地方购买的本书，可以访问http://www.packtpub.com/support并进行注册，相关的范例代码会直接用电子邮件发给你。

1.3.2　攻略小结

我们看到了若干Magic函数（所谓的“魔法函数”）的实际运用。这些函数以%字符开始。如果Magic函数只用作单行命令，可以选择省略%。

1.4　阅读手册页

进入IPython的pylab模式后，可以使用help命令查看NumPy中各种函数的手册页。不需要记住函数的完整名称，只需键入开头的几个字符，然后使用tab键自动补全函数名。作为示例，让我们看看arange函数的手册页有什么信息。

1.4.1　具体步骤

浏览可获得的信息，有以下两种方法。

- **调用help函数**

使用help命令，键入函数名的前面几个字符，然后按tab键：

```
In [1]: help ar
arange      arcsin      arctan2     argmin      around      array_equal array_split
arccos      arcsinh     arctanh     argsort     array       array_equiv array_str
arccosh     arctan      argmax      argwhere    array2string array_repr arrow
```

- **使用问号查询**

另一种方法是在函数名后面添加一个问号。你需要知道函数的完整名称，但不再需要键入help了。

```
In [3]: arange?
```

1.4.2　攻略小结

代码补全功能依赖于readline，你要确信已经安装了readline。使用问号查询方式，你看到的是docstrings中的信息。

1.5 安装 Matplotlib

Matplotlib是一个非常有用的绘图库，下一篇攻略就将用到它。Matplotlib依赖于NumPy的存在，但十有八九你已安装了NumPy。

具体步骤

本节将介绍怎样在Windows、Linux和Mac环境中安装Matplotlib，以及怎样从源文件安装。

▶ 在Windows环境中安装Matplotlib

可以使用Enthought发行版进行安装，详见http://www.enthought.com/products/epd.php。

可能需要把msvcp71.dll文件放到C:\Windows\system32目录下。可以访问http://www.dll-files.com/dllindex/dll-files.shtml?msvcp71下载这个dll文件。

▶ 在Linux环境中安装Matplotlib

让我们看看怎样在各种Linux发行版本中安装Matplotlib。

- Debian和Ubuntu版本的安装命令如下：

```
sudo apt-get install python-matplotlib
```

- Fedora/Redhat版本的安装命令如下：

```
su - yum install python-matplotlib
```

▶ 从源文件安装

从Sourceforge（http://sourceforge.net/projects/matplotlib/files/）下载最新的tar.gz格式的源文件，或者使用如下命令直接从Git版本仓库下载。

```
git clone git://github.com/matplotlib/matplotlib.git
```

下载完毕后，像往常一样使用如下命令构建和安装Matplotlib：

```
cd matplotlib
python setup.py install
```

▶ 在Mac环境中安装Matplotlib

从http://sourceforge.net/projects/matplotlib/files/matplotlib/获取最新的DMG文件并进行安装。

1.6　运行基于 Web 的 notebook

新近的IPython版本增加了一个令人兴奋的新特性——基于Web的notebook。一个被称为notebook服务器的程序可以通过Web方式提供notebook界面。现在我们可以启动一个notebook服务器，获得一个基于Web的IPython运行环境。notebook除了具备常规IPython环境中的大多数特性，还包括以下新特性。

- 显示图像和嵌入式图表
- 在文本单元格中使用HTML和Markdown
- notebook的导入和导出

1.6.1　准备工作

首先要确保已经安装了所有必需的软件。notebook依赖于tornado和zmq的存在。具体请参见1.2节。

1.6.2　具体步骤

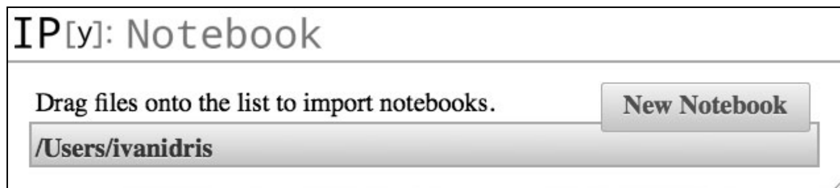
▶ 运行notebook

键入如下命令，启动一个notebook。

```
$ ipython notebook

[NotebookApp] Using existing profile dir: u'/Users/ivanidris/.
ipython/profile_default'
[NotebookApp] The IPython Notebook is running at:
http://127.0.0.1:8888
[NotebookApp] Use Control-C to stop this server and shut down
all kernels.
```

如你所见，我们用的是默认配置。notebook服务器运行在本地主机的8888端口。本章后续会介绍怎么修改这些默认的设置。notebook会在本机的默认浏览器中打开，也可以配置使用其他浏览器。



IPython会列出当前目录下所有的notebook文件。上图显示，当面目录下没有notebook文件。使用快捷键Ctrl+C可以停止notebook服务器的运行。

▶ 用pylab模式运行notebook

用pylab模式运行notebook，使用如下命令：

```
$ ipython notebook --pylab
```

这样可以自动加载SciPy、NumPy和Matplotlib模块。

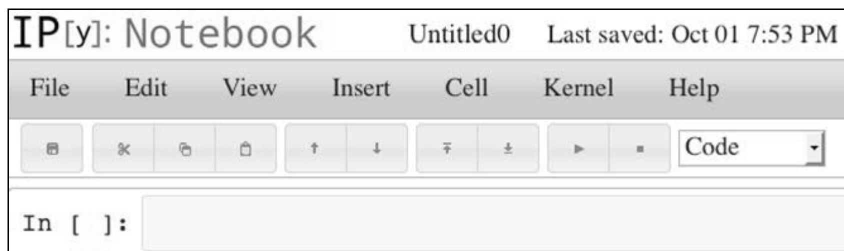
▶ 运行notebook时使用嵌入式图表

使用`inline`指令可以在notebook的单元格中显示嵌入式图表，命令如下。

```
$ ipython notebook --pylab inline
```

1. 创建一个notebook文件

点击New Notebook按钮，创建一个新的notebook文件。



2. 创建一个数组

使用`arange`函数创建一个数组。键入如下命令并按下Shift+Enter键。

```
In [1]: a = arange(7)
```

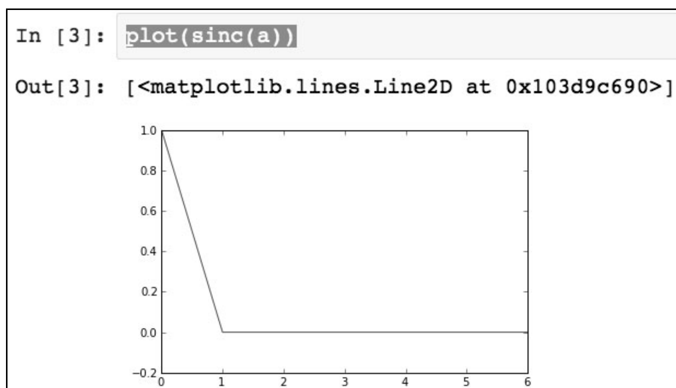
键入如下命令并按下Shift+Enter键，可以在Out[2]单元格中看到输出结果。

```
In [2]: a

Out[2]: array([0, 1, 2, 3, 4, 5, 6])
```

3. 绘制`sinc`函数

把数组a作为`sinc`函数的参数，其结果图示如下：



1.6.3 攻略小结

使用inline选项，Matplotlib绘制的图表将直接显示在输出单元格中。结合使用pylab模式，就不需要手动引入SciPy、NumPy和Matplotlib模块。

1.6.4 参考阅读

- 1.2节“安装IPython”

1.7 导出基于 Web 的 notebook

有时你需要和朋友或同事交换notebook中的内容。基于Web的notebook提供了几种导出数据的方法。

具体步骤

notebook的导出方式有以下几种：

- 打印选项

Print按钮并不是真的用来打印notebook中的内容，而是允许你以PDF或HTML文档的形式输出notebook的内容。

- 下载notebook

使用Download按钮，可以下载notebook中的内容到指定位置。可以指定下载内容的保存格式为.py文件（常规的Python程序）或者.ipynb文件（JSON格式）。把前一篇攻略中创建的notebook导出为.ipynb文件后，其内容如下所示：

```
{
 "metadata": {
  "name": "Untitled1"
 },
 "nbformat": 2,
 "worksheets": [
  {
    "cells": [
    {
      "cell_type": "code",
      "collapsed": false,
      "input": [
        "plot(sinc(a))"
      ],
      "language": "python",
      "outputs": [
      {
        "output_type": "pyout",
        "prompt_number": 3,
        "text": [
          "[&lt;matplotlib.lines.Line2D at
            0x103d9c690&gt;]"
        ]
      },
      {
        "output_type": "display_data",
        "png": "iVBORw0KGgoAAAANSUhEUgAAAXk
          AAAD9CAYAAABZVQdHAAAABHNCSVQICAgIf...
          mgkAAAAASUVORK5CYII=\n"
      }
      ],
      "prompt_number": 3
    }
    ]
  }
  ]
}
```



简明起见，上面这个文件的部分内容已被省略。该文件不是用来编辑或阅读的，但如果忽略其中表示图像的部分，还是很容易读懂的。更多有关JSON的信息请参见https://en.wikipedia.org/wiki/JSON。

▶ 保存notebook

使用Save按钮，notebook的内容将自动导出到一个JSON格式的.ipynb文件中。该文件会被存储到你启动IPython的目录。

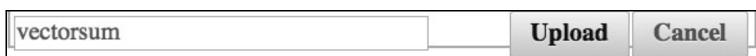
1.8　导入基于 Web 的 notebook

Python脚本可以导入到基于Web的notebook中，以前导出的notebook文件显然也可以导入到当前的notebook中。

具体步骤

把Python脚本导入到notebook中的步骤如下。

1. 把Python脚本从文件资源管理器（Explorer或Finder）拖曳到notebook界面上。vectorsum.py文件（*NumPy Beginner's Guide*中的例程）被拖曳到notebook界面后的屏幕截图如下所示：



2. 点击Upload按钮导入该脚本程序。如下图所示，IPython可以很好地完成导入工作，但不幸的是所有的代码都被放置在了一个单元格中，至少在本书写作的时候还是这样的状况。

```
File   Edit   View   Insert   Cell   Kernel   Help

In [ ]: #!/usr/bin/env/python

        import sys
        from datetime import datetime
        import numpy

        """
         Chapter 1 of NumPy Beginners Guide.
         This program demonstrates vector addition the Python way.
         Run from the command line as follows

          python vectorsum.py n

         where n is an integer that specifies the size of the vectors.

         The first vector to be added contains the squares of 0 up to n.
         The second vector contains the cubes of 0 up to n.
         The program prints the last 2 elements of the sum and the elapsed time.
        """

        def numpysum(n):
           a = numpy.arange(n) ** 2
           b = numpy.arange(n) ** 3
           c = a + b
```

3. 给脚本添加标签，使其能显示在多个单元格中。

为了能把代码分开放置到多个单元格中，需要使用特别的标签。这些标签实际上是Python的注释，看上去有点像XML标签。需要在代码的起始位置添加如下标签：

```
# <nbformat>2</nbformat>
```

该标签指明了notebook标签格式的版本号[①]。每个新的代码单元都用如下标签进行标记：

```
# <codecell>
```

标记后的代码如下所示：

```
# <nbformat>2</nbformat>
#!/usr/bin/env/python

from datetime import datetime
import numpy

"""
 NumPy Beginner's Guide第1章例程
 本程序演示了在Python中怎样实现向量的加法。
 在命令行界面中运行如下命令:

   python vectorsum.py n

 n是整数，表示向量的长度。

 第1个向量中放的是0到n的平方。
 第2个向量中放的是0到n的立方。
 程序会显示加法运算结果中的最后2个向量元素和该计算过程所耗费的时间。
"""

def numpysum(n):
   a = numpy.arange(n) ** 2
   b = numpy.arange(n) ** 3
   c = a + b

   return c

def pythonsum(n):
   a = range(n)
   b = range(n)
   c = []

   for i in range(len(a)):
       a[i] = i ** 2
       b[i] = i ** 3
       c.append(a[i] + b[i])

   return c
```

① 较新版本的notbook使用的版本号是3，版本号为2的脚本可以被自动转换为最新版本。——译者注

```
# <codecell>
size = int(50)

# <codecell>
start = datetime.now()
c = pythonsum(size)
delta = datetime.now() - start
print "The last 2 elements of the sum", c[-2:]
print "PythonSum elapsed time in microseconds", delta.microseconds

# <codecell>
start = datetime.now()
c = numpysum(size)
delta = datetime.now() - start
print "The last 2 elements of the sum", c[-2:]
print "NumPySum elapsed time in microseconds", delta.microseconds
```

依据标签所在的位置，代码被分解到了多个单元格中，如下图所示。

```
In [ ]: size = int(50)

In [ ]: start = datetime.now()

        c = pythonsum(size)

        delta = datetime.now() - start

        print "The last 2 elements of the sum", c[-2:]

        print "PythonSum elapsed time in microseconds", delta.microseconds

In [ ]: start = datetime.now()

        c = numpysum(size)

        delta = datetime.now() - start

        print "The last 2 elements of the sum", c[-2:]

        print "NumPySum elapsed time in microseconds", delta.microseconds
```

1.9　配置 notebook 服务器

需要考虑位于公共域中的notebook服务器的安全性。为此，需要设置密码和使用SSL证书来建立连接。我们需要用证书实现基于https的安全通信。更多相关信息请见 https://en.wikipedia.org/wiki/Transport_Layer_Security。

1.9.1 具体步骤

下述步骤介绍了怎样配置安全的notebook服务器。

1. 生成密码。可以在IPython环境中生成一个密码。启动一个新的IPython会话，键入如下命令：

```
In [1]: from IPython.lib import passwd


In [2]: passwd()
Enter password:
Verify password:
Out[2]: 'sha1:0e422dfccef2:84cfbcb
  b3ef95872fb8e23be3999c123f862d856'
```

输入第二行命令后，系统会提示你输入密码。你需要记住这个密码。之后会输出一个长字符串。复制该字符串，后面我们要用到它。

2. 创建SSL证书。为了能创建SSL证书，你需要让openssl命令位于可访问路径中。

openssl命令行工具的设置过程不算太复杂，但可能有一些棘手的细节。遗憾的是，这些内容超出了本书的范围，好在网上有很多教程可以为你提供帮助。

执行如下命令，生成一个文件名为mycert.pem的证书：

```
$ openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout
mycert.pem -out mycert.pem
Generating a 1024 bit RSA private key
......++++++
........................++++++
writing new private key to 'mycert.pem'
-----You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:
Email Address []:
```

openssl会提示你在一些区域里填写内容，请参阅相关的手册页了解更多信息。

3. 创建一个服务器配置。使用如下命令为服务器创建一个专用的配置：

```
ipython profile create nbserver
```

4. 编辑配置文件。对配置文件进行编辑。在本例中，配置文件所在的位置是~/.ipython/profile_nbserver/ipython_notebook_config.py。

这个配置文件包括很多行内容，这里就不列出了。我们至少需要修改如下几行内容：

```
c.NotebookApp.certfile = u'/absolute/path/to/your/certificate'
c.NotebookApp.password = u'sha1:b...your password'
c.NotebookApp.port = 9999
```

注意我们做的改动是，把证书位置指向了我们刚才创建的SSL证书，设置了密码，并且把端口号改为9999。

5. 启动服务器。使用如下命令启动服务器，检查所做的修改是否已生效。

```
ipython notebook --profile=nbserver

[NotebookApp] Using existing profile dir: u'/Users/ivanidris/.
ipython/profile_nbserver'

[NotebookApp] The IPython Notebook is running at:
https://127.0.0.1:9999

[NotebookApp] Use Control-C to stop this server and shut down
all kernels.
```

服务器已运行在9999端口，你需要使用https方式连接到该服务器。如果一切顺利，我们会看到一个登录页面。你很可能还需要接受浏览器发来的一个安全例外警告。

1.9.2 攻略小结

我们为公共域服务器创建了一个专用的配置。IPython中已有一些配置样本存在，例如默认配置。创建配置时，会在.ipython目录下添加一个profile_<profilename>文件夹，文件夹中包括配置文件等内容。通过使用--profile=<profile_name>命令行选项，可以加载指定的配置。用如下命令可以列出已经存在的配置。

```
ipython profile list
```

```
Available profiles in IPython:
    cluster
    math
    pysh
    python3

    The first request for a bundled profile will copy it
    into your IPython directory (/Users/ivanidris/.ipython),
    where you can customize it.

Available profiles in /Users/ivanidris/.ipython:
    default
    nbserver
    sh
```

1.10　初探 SymPy 配置

IPython中包含一个SymPy配置的样本。SymPy是一个Python的符号计算库。例如，我们可以利用SymPy来化简代数表达式或做微分运算，其功能类似于Mathematica和Maple。显然SymPy是一个有趣的软件，但它不是我们学习NumPy的过程中必须要了解的内容。可以认为本节是一个可选的、有奖励性质的攻略。就像餐后甜点一样，你可以放心地跳过本节，虽然这样做有可能让你错过本章最美妙的一段内容。

1.10.1　准备工作

使用easy_install或pip安装SymPy：

```
easy_install sympy
sudo pip install sympy
```

1.10.2　具体步骤

1. 浏览配置文件，其所在位置是~/.ipython/profile_sympy/ipython_config.py。具体内容如下。

```
c = get_config()
app = c.InteractiveShellApp

# This can be used at any point in a config file to load a sub
config
# and merge it into the current one.
load_subconfig('ipython_config.py', profile='default')

lines = """
from __future__ import division
```

```
from sympy import *
x, y, z, t = symbols('x y z t')
k, m, n = symbols('k m n', integer=True)
f, g, h = symbols('f g h', cls=Function)
"""

# You have to make sure that attributes that are containers
already
# exist before using them.  Simple assigning a new list will
override
# all previous values.

if hasattr(app, 'exec_lines'):
    app.exec_lines.append(lines)
else:
    app.exec_lines = [lines]

# Load the sympy_printing extension to enable nice printing of
sympy expr's.
if hasattr(app, 'extensions'):
    app.extensions.append('sympyprinting')
else:
    app.extensions = ['sympyprinting']
```

上面这段代码实现的功能是：

- 加载默认配置
- 导入SymPy包
- 定义符号

2. 使用SymPy配置选项启动IPython，命令如下。

```
ipython --profile=sympy
```

3. 使用下图所示命令，展开一个代数表达式。

```
In [1]: expand((x+y)**7)
Out[1]:

 7      6         5  2       4  3       3  4       2  5         6    7
x  + 7⋅x ⋅y + 21⋅x ⋅y  + 35⋅x ⋅y  + 35⋅x ⋅y  + 21⋅x ⋅y  + 7⋅x⋅y  + y
```

第2章 高级索引和数组概念

本章主要内容：

- ❑ 安装SciPy
- ❑ 安装PIL
- ❑ 调整图像大小
- ❑ 对比视图和副本
- ❑ 翻转图像
- ❑ 高级索引
- ❑ 位置列表型索引
- ❑ 布尔型索引
- ❑ 数独游戏中的跨度技巧
- ❑ 用广播机制扩展数组

2.1 引言

NumPy以高效率的数组著称。这个美誉要部分归因于索引的易用性。我们将以图像处理为例展示高级的索引技巧。在深入研究索引之前，我们要先安装必需的软件——SciPy和PIL。



在本书的官网http://www.packtpub.com上，可以找到本章中攻略的源代码。你也可以访问http://www.packtpub.com/support并进行注册，相关的文件会直接用电子邮件发给你。

本章中的一些实例涉及图像处理。为此，我们需要用到Python图像库（PIL）。不用着急，本章后续部分会适时给出指导和建议，帮助你安装PIL和其他需要的Python软件。

2.2　安装 SciPy

SciPy是一个和NumPy密切相关的Python科学计算库。实际上，很多年前，SciPy和NumPy归属于同一个项目。本节将介绍怎样安装SciPy。

2.2.1　准备工作

在第1章的1.2一节，我们介绍了怎样安装setuptools和pip。如有必要，请再次阅读该攻略。

2.2.2　具体步骤

下面介绍安装SciPy的具体步骤。

▶ 从源文件安装

如果已经安装了Git，可以使用如下命令复制SciPy的版本库：

```
git clone https://github.com/scipy/scipy.git

python setup.py build
python setup.py install --user
```

上面这两条命令把SciPy安装到了你的用户目录，要求Python的版本不低于2.6。

构建安装文件之前，你还需要安装以下依赖包：

- BLAS和LAPACK库
- C和Fortran编译器

作为NumPy安装文件的一部分，你有可能已经安装好这些软件了。

▶ 在Linux环境中安装SciPy

大多数Linux发行版都有SciPy安装包。我们将针对几个流行的Linux发行版，介绍必要的安装步骤。

- 在Red Hat、Fedora和CentOS中安装SciPy，需要在命令行界面中运行如下命令：

  ```
  yum install python-scipy
  ```

- 在Mandriva中安装SciPy，需要运行如下命令行指令：

  ```
  urpmi python-scipy
  ```

- 在Gentoo中安装SciPy，需要运行如下命令行指令：

```
sudo emerge scipy
```

- 在Debian或Ubuntu中安装时，需要键入如下命令：

```
sudo apt-get install python-scipy
```


▶ 在Mac OS X环境中安装SciPy

需要用到苹果开发工具Xcode，因为其中包含了BLAS和LAPACK库。可以在苹果应用商店或者Mac电脑附带的DVD安装光盘中找到Xcode，还可以访问苹果开发者网站的 https://developer.apple. com/ technologies/tools/页面获取最新版本。要确保Xcode中所有的可选包都已安装。

你可能在安装NumPy的时候已经安装好了Fortran编译器。在http://r.research.att.com/tools/中可以找到gfortran的二进制安装文件。

▶ 使用easy_install或pip安装SciPy

使用如下两条命令之一完成安装。

```
sudo pip install scipy
```

```
easy_install scipy
```

▶ 在Windows环境中安装

如果你已经安装了Python，推荐的安装方法是下载和使用二进制的SciPy安装文件。另一种选择是安装Enthought公司的Python发行版，里面包括了SciPy在内的多种Python科学计算软件包。

▶ 检查安装情况

使用如下代码，检查SciPy的安装情况：

```
import scipy
print scipy.__version__
print scipy.__file__
```

这将显示正确的SciPy版本号。

2.2.3 攻略小结

大多数包管理器会替你处理各种依赖关系。但有些时候，你需要手动安装这些依赖文件。很遗憾，这些内容超出了本书的讨论范围。如果你碰到了问题，可以到下面两个地方寻求帮助：

- freenode的#scipy IRC频道
- 位于http://www.scipy.org/Mailing_Lists的SciPy邮件列表

2.3　安装 PIL

PIL是Python的图像库。本章后续和图像处理有关的攻略都要用到它，因此需要提前准备好。

具体步骤

让我们看看怎样安装PIL。

- 在Windows中安装PIL

从PIL的官网http://www.pythonware.com/products/pil/下载Windows版本的可执行安装文件，并完成安装。

- 在Debian或Ubuntu中安装

在Debian或Ubuntu中，使用如下命令安装PIL。

```
sudo apt-get install python-imaging
```

- 使用easy_install或pip安装

写作本书的时候，Red Hat、Fedora和CentOS中的包管理器似乎尚未直接支持PIL的安装。因此，如果你使用的是这些Linux发行版中的某一个，请参照下面的步骤。

使用如下两条命令都可以完成安装。

```
easy_install PIL
sudo pip install PIL
```

2.4　调整图像大小

本攻略将把SciPy库中的一个示例图像Lena，加载到一个数组中。顺便说一下，本章不是要讨论图像处理，我们只是把图像数据用作输入。



> Lena Soderberg是1972年出现在《花花公子》杂志上的一个人物。因为历史的原因，图像处理领域中会经常用到她的一幅肖像。请勿多虑，在工作中使用这个肖像是绝对安全的。

我们将用repeat函数调整该图像的大小。该函数用来对一个数组进行repeat操作，其实际效果就是按一定比例调整图像大小。

2.4.1 准备工作

使用本攻略的前提条件是SciPy、Matplotlib和PIL都已安装好了。请参见本章和上一章中的相关攻略。

2.4.2 具体步骤

1. 加载图像Lena到数组中。

SciPy中有一个lena函数，可以用该函数把图像Lena加载到NumPy数组中。

```
lena = scipy.misc.lena()
```

SciPy的0.10及之后的版本重构了部分代码，如果你用的是重构之前的旧版本，正确的代码是：

```
lena = scipy.lena()
```

2. 检查数组的形状。

用numpy.testing包中的assert_equal函数检查数组Lena的形状，这是一个可选的完整性检查。

```
numpy.testing.assert_equal((LENA_X, LENA_Y), lena.shape)
```

3. 调整数组Lena的大小。

用repeat函数调整数组Lena的大小。需要分别在x和y方向给出一个调整系数：

```
resized = lena.repeat(yfactor, axis=0).repeat(xfactor, axis=1)
```

4. 绘制数组对应的图像。

我们将把原始图像Lena和调整大小后的图像，分别画到一个网格内的两个子图中。在第一个子图中绘制数组Lena。

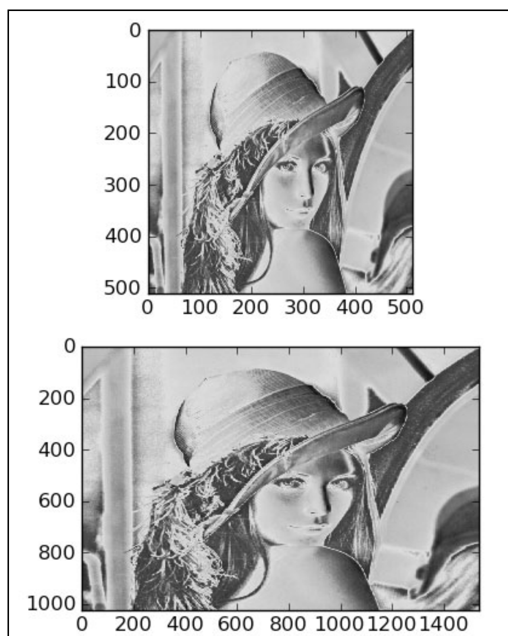
```
matplotlib.pyplot.subplot(211)
matplotlib.pyplot.imshow(lena)
```

Matplotlib中的subplot函数用来创建子图。这个函数以一个3位的整数作为参数，最高位的数字代表行数，第二个数字代表列数，最后一个数字代表子图的位置索引（从1开始编号）。imshow函数用来显示图像，show函数用来显示最终的结果。

把大小调整后的数组绘制在第二个子图中，并显示最终的结果。该子图的位置索引是2。

```
matplotlib.pyplot.subplot(212)
matplotlib.pyplot.imshow(resized)
matplotlib.pyplot.show()
```

最终的结果如下图所示，第一个子图是原始图像，第二个子图是大小调整后的图像。



本攻略的完整代码如下。

```
import scipy.misc
import sys
import matplotlib.pyplot
import numpy.testing

# 这个脚本用来调整SciPy库中图像Lena的大小。

if(len(sys.argv) != 3):
   print "Usage python %s yfactor xfactor" % (sys.argv[0])
   sys.exit()

# 加载图像Lena到一个数组中
lena = scipy.misc.lena()
# 图像Lena的规格
LENA_X = 512
LENA_Y = 512
# 检查数组Lena的形状
numpy.testing.assert_equal((LENA_X, LENA_Y), lena.shape)

# 获取调整系数
yfactor = float(sys.argv[1])
```

```
xfactor = float(sys.argv[2])

# 调整数组Lena的大小
resized = lena.repeat(yfactor, axis=0).repeat(xfactor, axis=1)

# 检查大小调整后的数组的形状
numpy.testing.assert_equal((yfactor * LENA_Y, xfactor * LENA_Y),
resized.shape)

# 绘制数组Lena
matplotlib.pyplot.subplot(211)
matplotlib.pyplot.imshow(lena)

# 绘制大小调整后的数组
matplotlib.pyplot.subplot(212)
matplotlib.pyplot.imshow(resized)
matplotlib.pyplot.show()
```


2.4.3 攻略小结

`repeat`函数对数组进行repeat操作。对于本例的情况，其结果是改变了原始图像的大小。Matplotlib中的`subplot`函数用来创建子图，`imshow`函数用来显示图像，`show`函数用来显示最终的结果。

2.4.4 参考阅读

- 1.5节“安装Matplotlib”
- 2.2节“安装SciPy”
- 2.3节“安装PIL”

2.5 创建视图和副本

有一个重要的事情我们必须要弄清楚，即我们是在与一个共享的数组视图打交道，还是获得了数组数据的一个副本。例如，一个切片（slice）对应一个视图。这意味着，如果你把一个切片赋值给一个变量，随后改变了切片所在数组中的内容，那么这个变量的值也会改变。我们将用著名的图像Lena创建一个数组，复制该数组，创建一个视图，最后再修改这个视图。

2.5.1 准备工作

必须具备的条件和上一个攻略相同。

2.5.2　具体步骤

创建数组Lena的一个副本和视图。

1. 创建数组Lena的一个副本：

```
acopy = lena.copy()
```

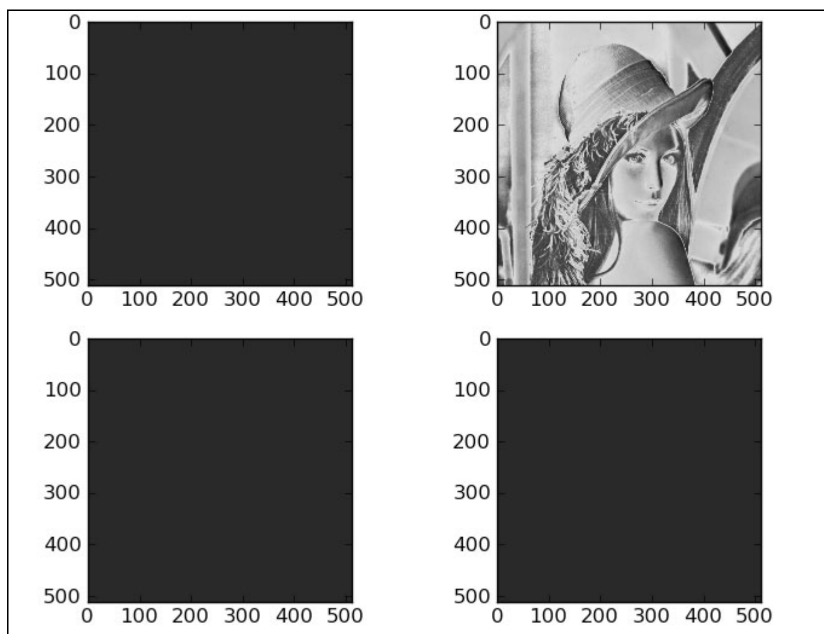
2. 创建数组Lena的一个视图：

```
aview = lena.view()
```

3. 利用flat迭代器，把视图中所有的值清零：

```
aview.flat = 0
```

最终结果是，只有一幅图像显示的是Lena的肖像，其他几幅图像都被完全删剪掉了。



本攻略展示了数组的视图和副本的不同之处，完整代码如下。

```
import scipy.misc
import matplotlib.pyplot

lena = scipy.misc.lena()
acopy = lena.copy()
aview = lena.view()
```

```
# 绘制数组Lena
matplotlib.pyplot.subplot(221)
matplotlib.pyplot.imshow(lena)

# 绘制数组Lena的副本
matplotlib.pyplot.subplot(222)
matplotlib.pyplot.imshow(acopy)

# 绘制数组Lena的视图
matplotlib.pyplot.subplot(223)
matplotlib.pyplot.imshow(aview)

# 改变视图内容后再绘制视图
aview.flat = 0
matplotlib.pyplot.subplot(224)
matplotlib.pyplot.imshow(aview)

matplotlib.pyplot.show()
```

2.5.3 攻略小结

如你所见，在程序的后半段，通过改变视图的内容，我们改变了原始数组Lena中的内容。这导致有三幅图像会显示为单一的蓝色（如果你只能看到黑白图像，将显示为黑色），只有原始数组的副本不受视图内容改变的影响。一定要牢记，视图不是只读的。

2.6 翻转图像

作为一个演示实例，我们将对SciPy库中的图像Lena进行翻转操作（当然是出于科学研究的需要）。除了对图像进行翻转，还会绘制出它的一部分以及对其应用遮罩的效果。

2.6.1 具体步骤

具体步骤如下。

1. 绘制翻转后的图像。

使用如下代码，沿纵轴方向对数组Lena进行翻转操作：

```
matplotlib.pyplot.imshow(lena[:,::-1])
```

2. 绘制图像的一部分。

把图像的一部分切割出来并进行显示。我们需要先查看一下数组Lena的形状（shape）属性。

shape是一个用来表示数组维数的元组。如下代码实际上是把图像Lena左上角的四分之一选取并显示出来了。

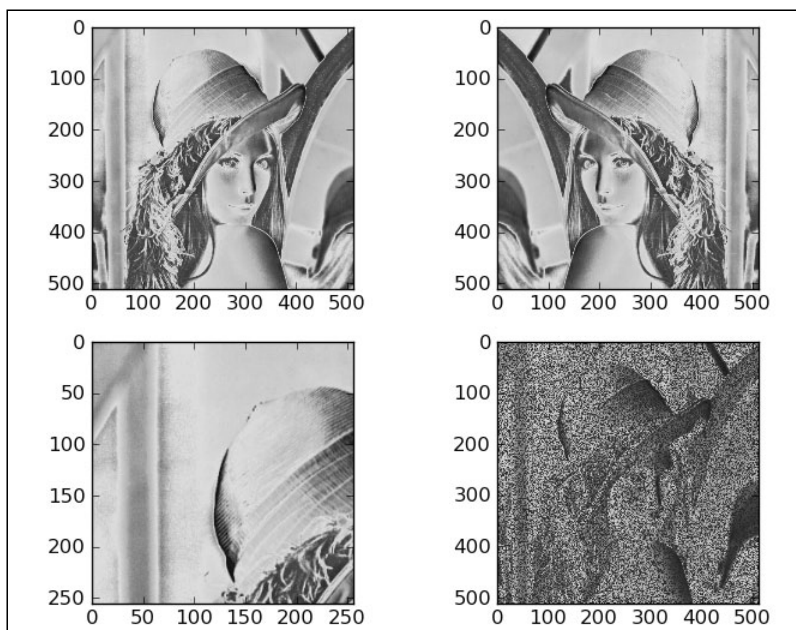
```
matplotlib.pyplot.imshow(lena[:lena.shape[0]/2,
  :lena.shape[1]/2])
```

3. 对图像应用遮罩的效果。

找到数组Lena中所有的偶数并把它们置零（这只是出于演示目的随意选定的标准），这样就实现了对图像的遮罩效果。获得数组的一个副本，并把其中的偶数置零，其效果相当于在图像上画了很多蓝色的点（如果你看到的是黑白图像，那只能看到黑点）。

```
mask = lena % 2 == 0
masked_lena = lena.copy()
masked_lena[mask] = 0
```

所有这些操作的结果是生成一个2 × 2的图像网格，如下图所示。



本攻略的完整代码如下。

```
import scipy.misc
import matplotlib.pyplot

# 加载数组Lena
lena = scipy.misc.lena()
```

```
# 绘制数组Lena
matplotlib.pyplot.subplot(221)
matplotlib.pyplot.imshow(lena)

# 绘制翻转后的数组
matplotlib.pyplot.subplot(222)
matplotlib.pyplot.imshow(lena[:,::-1])

# 绘制数组的一部分
matplotlib.pyplot.subplot(223)
matplotlib.pyplot.imshow(lena[:lena.shape[0]/2,:lena.shape[1]/2])

# 应用遮罩的效果
mask = lena % 2 == 0
masked_lena = lena.copy()
masked_lena[mask] = 0
matplotlib.pyplot.subplot(224)
matplotlib.pyplot.imshow(masked_lena)

matplotlib.pyplot.show()
```

2.6.2 参考阅读

- 1.5节“安装Matplotlib”
- 2.2节“安装SciPy”
- 2.3节“安装PIL”

2.7 高级索引

本攻略将使用高级索引技术，把图像Lena对角线位置上的数值置零，其效果就是在图像上画出两条交叉的蓝色（或者是黑色）对角线。给图像画叉没什么特别的意思，只是练习的需要。与常规索引不同的是，高级索引不使用整数或切片作为索引值。

2.7.1 具体步骤

从画第一条对角线开始，步骤如下。

1. 把第一条对角线位置上的数值置零。

为了能把对角线上的值置零，需要把数组的两个索引值x和y定义为不同的整数列表：

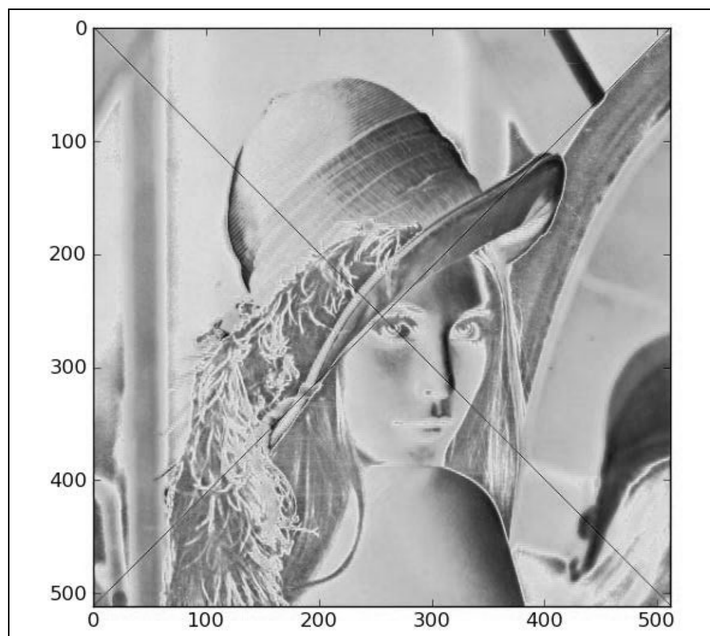
```
lena[range(xmax), range(ymax)] = 0
```

2. 把另一条对角线位置上的数值置零。

为了能把另一条对角线上的值置零，需要定义另外一对整数列表，但基本原理是一样的：

```
lena[range(xmax-1,-1,-1), range(ymax)] = 0
```

在最终得到的图像上，可以看到两条交叉的对角线，如下图所示。



本攻略的完整代码如下。

```
import scipy.misc
import matplotlib.pyplot

# 通过把对角线上的值置零，演示了高级索引的用法.

# 加载数组Lena
lena = scipy.misc.lena()
xmax = lena.shape[0]
ymax = lena.shape[1]

# 高级索引
# 把对角线位置上的值置零
# x 0-xmax
# y 0-ymax
lena[range(xmax), range(ymax)] = 0

# 把另一条对角线位置上的值置零
```

```
# x xmax-0
# y 0-ymax
lena[range(xmax-1,-1,-1), range(ymax)] = 0

# 绘制对角线置零后的图像Lena
matplotlib.pyplot.imshow(lena)
matplotlib.pyplot.show()
```


2.7.2 攻略小结

我们把数组的索引值x和y分别定义为单独的整数列表。这两个整数列表被用作数组Lena的索引。高级索引的实现基于一个内部的NumPy迭代器对象，具体执行过程包括三个步骤。

1. 创建迭代器对象。

2. 把迭代器对象绑定到数组。

3. 通过迭代器访问数组元素。

2.8 位置列表型索引

让我们使用ix_函数，把图像Lena搅乱。ix_函数能利用多个序列生成一个网状结构。

具体步骤

首先把数组的索引随机重排，步骤如下。

1. 把数组的索引随机重排。

利用numpy.random模块中的shuffle函数，生成一个随机排列的索引值数组：

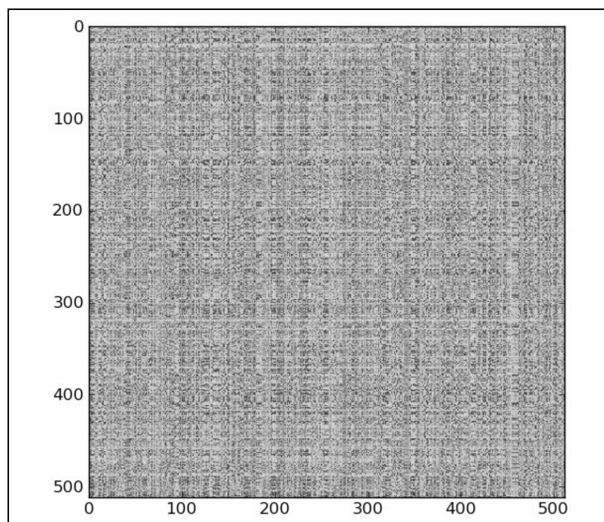
```
def shuffle_indices(size):
   arr = numpy.arange(size)
   numpy.random.shuffle(arr)

   return arr
```

2. 用随机重排后的索引画出数组。

```
matplotlib.pyplot.imshow(lena[numpy.ix_(xindices, yindices)])
```

最终得到的是一幅被完全弄乱的Lena肖像，如下图所示。



本攻略的完整代码如下。

```
import scipy.misc
import matplotlib.pyplot
import numpy.random
import numpy.testing

# 加载数组Lena
lena = scipy.misc.lena()
xmax = lena.shape[0]
ymax = lena.shape[1]

def shuffle_indices(size):
   arr = numpy.arange(size)
   numpy.random.shuffle(arr)

   return arr

xindices = shuffle_indices(xmax)
numpy.testing.assert_equal(len(xindices), xmax)
yindices = shuffle_indices(ymax)
numpy.testing.assert_equal(len(yindices), ymax)

# 绘制数组Lena
matplotlib.pyplot.imshow(lena[numpy.ix_(xindices, yindices)])
matplotlib.pyplot.show()
```

2.9　布尔型索引

布尔型索引就是基于布尔数组的索引，属于高级索引技术的范畴。

2.9.1 具体步骤

我们将把这种索引技术应用到图像处理过程。

1. 在图像上添加点状的对角线。

这个步骤和2.7节中画对角线的过程有点类似，只不过这次我们选取的是图像对角线上能被4整除的索引值对应的点。

```
def get_indices(size):
   arr = numpy.arange(size)
   return arr % 4 == 0
```

然后应用这个选择，并画出这些点。

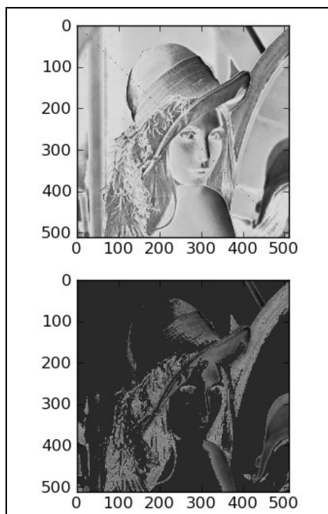
```
lena1 = lena.copy()
xindices = get_indices(lena.shape[0])
yindices = get_indices(lena.shape[1])
lena1[xindices, yindices] = 0
matplotlib.pyplot.subplot(211)
matplotlib.pyplot.imshow(lena1)
```

2. 把指定范围内的数值置零。

把数组中大小介于最大值的四分之一和最大值的四分之三之间的数值置零。

```
lena2[(lena > lena.max()/4) &
  (lena < 3 * lena.max()/4)] = 0
```

最终得到两幅新的图像，如下图所示。



本攻略的完整代码如下。

```
import scipy.misc
import matplotlib.pyplot
import numpy

# 加载数组Lena
lena = scipy.misc.lena()

def get_indices(size):
   arr = numpy.arange(size)
   return arr % 4 == 0

# 绘制图像Lena
lena1 = lena.copy()
xindices = get_indices(lena.shape[0])
yindices = get_indices(lena.shape[1])
lena1[xindices, yindices] = 0
matplotlib.pyplot.subplot(211)
matplotlib.pyplot.imshow(lena1)

lena2 = lena.copy()
# 选取最大值的1/4到最大值的3/4之间的数据
lena2[(lena > lena.max()/4) & (lena < 3 * lena.max()/4)] = 0
matplotlib.pyplot.subplot(212)
matplotlib.pyplot.imshow(lena2)

matplotlib.pyplot.show()
```

2.9.2　攻略小结

布尔型索引是高级索引的一种形式，因此它们的实现方式基本上是相同的。这意味着，布尔型索引也是利用一个特定的迭代器对象实现的。

2.9.3　参考阅读

- 2.7节“高级索引”

2.10　数独游戏中的跨度技巧

NumPy的ndarray类中有一个跨度（strides）属性域，实现为一个元组。在遍历数组时，用跨度来指明每一个维度上的步进值是多少字节。我们将巧妙运用跨度技巧，把数独谜题划分为3 × 3的九宫格形式。



对数独游戏规则的解释超出了本书的范围，请访问 http://en.wikipedia.org/wiki/Sudoku获取更多相关信息。

2.10.1 具体步骤

1. 定义数独谜题数组。

首先定义数独谜题数组，其内容就是一个真实的、已经解开的数独谜题。

```
sudoku = numpy.array([
    [2, 8, 7, 1, 6, 5, 9, 4, 3],
    [9, 5, 4, 7, 3, 2, 1, 6, 8],
    [6, 1, 3, 8, 4, 9, 7, 5, 2],
    [8, 7, 9, 6, 5, 1, 2, 3, 4],
    [4, 2, 1, 3, 9, 8, 6, 7, 5],
    [3, 6, 5, 4, 2, 7, 8, 9, 1],
    [1, 9, 8, 5, 7, 3, 4, 2, 6],
    [5, 4, 2, 9, 1, 6, 3, 8, 7],
    [7, 3, 6, 2, 8, 4, 5, 1, 9]
    ])
```

2. 计算跨度。

`ndarray`类的`itemsize`域指明数组中每个元素所占用的字节数。利用`itemsize`来计算跨度：

```
strides = sudoku.itemsize *
  numpy.array([27, 3, 9, 1])
```

3. 划分九宫格。

用numpy.lib.stride_tricks模块中的`as_strided`函数，把谜题数组划分为九宫格的形式：

```
squares = numpy.lib.stride_tricks.as_strided
  (sudoku, shape=shape, strides=strides)
print(squares)
```

这样就能把各个宫格分别打印出来。

```
[[[[2 8 7]
    [9 5 4]
    [6 1 3]]

  [[1 6 5]
    [7 3 2]
    [8 4 9]]
```

```
  [[9 4 3]
    [1 6 8]
    [7 5 2]]]


 [[[8 7 9]
    [4 2 1]
    [3 6 5]]

  [[6 5 1]
    [3 9 8]
    [4 2 7]]

 [[2 3 4]
   [6 7 5]
   [8 9 1]]]


[[[1 9 8]
   [5 4 2]
   [7 3 6]]

 [[5 7 3]
   [9 1 6]
   [2 8 4]]

 [[4 2 6]
   [3 8 7]
   [5 1 9]]]]
```

本攻略的完整代码如下。

```
import numpy

sudoku = numpy.array([
   [2, 8, 7, 1, 6, 5, 9, 4, 3],
   [9, 5, 4, 7, 3, 2, 1, 6, 8],
   [6, 1, 3, 8, 4, 9, 7, 5, 2],
   [8, 7, 9, 6, 5, 1, 2, 3, 4],
   [4, 2, 1, 3, 9, 8, 6, 7, 5],
   [3, 6, 5, 4, 2, 7, 8, 9, 1],
   [1, 9, 8, 5, 7, 3, 4, 2, 6],
   [5, 4, 2, 9, 1, 6, 3, 8, 7],
   [7, 3, 6, 2, 8, 4, 5, 1, 9]
   ])

shape = (3, 3, 3, 3)

strides = sudoku.itemsize * numpy.array([27, 3, 9, 1])
```

```
squares = numpy.lib.stride_tricks.as_strided
  (sudoku, shape=shape, strides=strides)
print(squares)
```


2.10.2　攻略小结

我们应用跨度技巧，把数独谜题划分为3 × 3的九宫格形式。跨度告诉我们，在遍历数独数组时，每一步需要跳过多少字节。

2.11　用广播机制扩展数组

你也许还不了解广播机制这个术语，就已经把它应用到数组运算过程中了。简而言之，NumPy总是试图完成运算，即便运算对象的形状并不匹配。本攻略中，我们将把一个数组和一个标量相乘。标量会被“扩展”为和数组对象相同的形状，然后与之相乘。我们将下载一个声音文件，然后为它创建一个音量减小的新版本。

具体步骤

从读取一个WAV文件开始，步骤如下。

1. 读取一个WAV文件。

我们使用标准的Python代码，下载一个声音文件，其内容是Austin Powers演唱的歌曲“Smashing, baby”。SciPy有一个wavefile模块，用来加载声音数据或生成WAV文件。如果SciPy已经安装，我们就可以使用该模块。`read`函数返回一个包含声音数据的数组和采样率。本例中我们只关心声音数据。

```
sample_rate, data = scipy.io.wavfile.read(WAV_FILE)
```

2. 画出原始的WAV数据。

用Matplotlib画出原始的WAV数据，并把这个子图的标题命名为Original。

```
matplotlib.pyplot.subplot(2, 1, 1)
matplotlib.pyplot.title("Original")
matplotlib.pyplot.plot(data)
```

3. 创建一个新的数组。

我们用NumPy创建一个音量减小的声音样本。其实就是把原始的声音数据数组与一个常数相乘，生成一个数据值减小的新数组，广播机制的魔力就体现在这里。最后，为了能生成WAV格式

的文件，我们需要确保新数组的数据类型与原始数组一致。

```
newdata = data * 0.2
newdata = newdata.astype(numpy.uint8)
```

4. 生成新的WAV文件。

把新数组写入到一个新的WAV文件中，如下所示。

```
scipy.io.wavfile.write("quiet.wav",
    sample_rate, newdata)
```

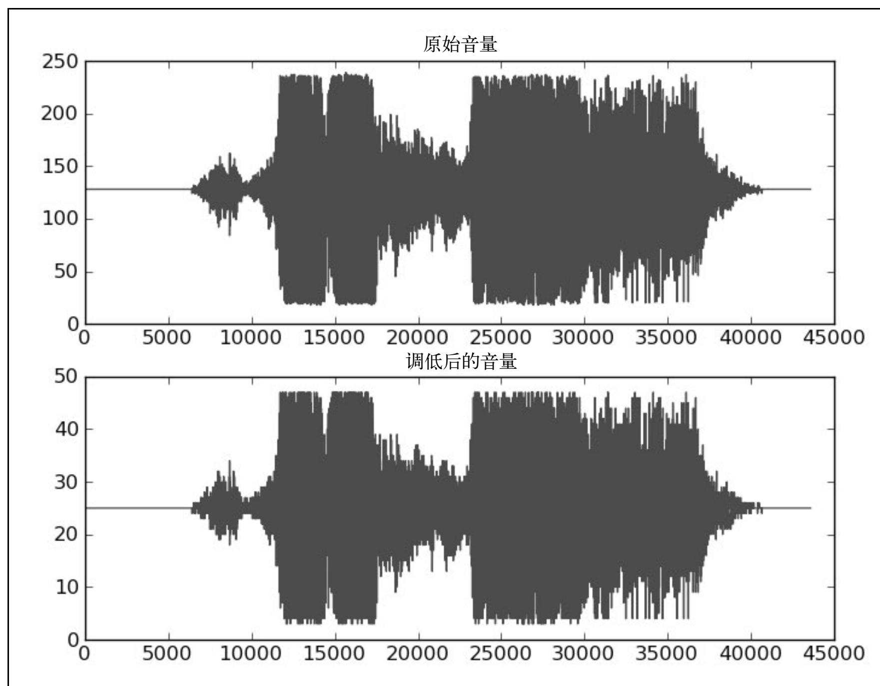
5. 画出新的WAV数据。

用Matplotlib画出新数组。

```
matplotlib.pyplot.subplot(2, 1, 2)
matplotlib.pyplot.title("Quiet")
matplotlib.pyplot.plot(newdata)

matplotlib.pyplot.show()
```

最终结果如下图所示，两个子图分别对应原始的WAV文件和数值按比例缩小后的新数组。



本攻略的完整代码如下。

```
import scipy.io.wavfile
import matplotlib.pyplot
import urllib2
import numpy

response = urllib2.urlopen('http://www.thesoundarchive.com/
austinpowers/smashingbaby.wav')
print response.info()
WAV_FILE = 'smashingbaby.wav'
filehandle = open(WAV_FILE, 'w')
filehandle.write(response.read())
filehandle.close()
sample_rate, data = scipy.io.wavfile.read(WAV_FILE)
print "Data type", data.dtype, "Shape", data.shape

matplotlib.pyplot.subplot(2, 1, 1)
matplotlib.pyplot.title("Original")
matplotlib.pyplot.plot(data)

newdata = data * 0.2
newdata = newdata.astype(numpy.uint8)
print "Data type", newdata.dtype, "Shape", newdata.shape

scipy.io.wavfile.write("quiet.wav",
    sample_rate, newdata)

matplotlib.pyplot.subplot(2, 1, 2)
matplotlib.pyplot.title("Quiet")
matplotlib.pyplot.plot(newdata)

matplotlib.pyplot.show()
```

第3章 常用函数

本章将介绍一些常用的函数：

- sqrt、log、arange、astype和sum
- ceil、modf、where、ravel和take
- sort和outer
- diff、sign和eig
- histogram和polyfit
- compress和randint

我们将通过以下攻略讨论这些函数的用法：

- 斐波那契数列求和
- 寻找质因数
- 寻找回文数
- 确定稳态向量
- 探索幂律分布
- 定期在低点做交易
- 模拟在随机时间点做交易
- 用埃式筛筛选整数

3.1 引言

本章将介绍常用函数。这些函数我们差不多每天都会用到，显然，具体用到哪些函数因人而异。NumPy提供了数量众多的函数，几乎不可能去全部了解，但本章中介绍的这些函数是我们必须要熟悉的。你可以从本书官网http://www.packtpub.com下载本章的源代码。

3.2　斐波那契数列求和

本攻略中，我们将对斐波那契数列中取值不大于四百万且为偶数的项进行求和运算。斐波那契数列是从0开始的一个整数序列，除了第一项和第二项是0和1之外，其余各项的取值都是该项之前的两项求和的结果。[①]

> 有关斐波那契数列的更多信息，请参阅维基百科页面http://en.wikipedia.org/wiki/Fibonacci_number的相关介绍。

本攻略使用了一个基于黄金比例（golden ratio）的公式。黄金比例就是一个无理数，有着类似于π的独特属性。我们将用到`sqrt`、`log`、`arange`、`astype`和`sum`函数。

3.2.1　具体步骤

首先需要计算黄金比例（http://en.wikipedia.org/wiki/Golden_ratio）。黄金比例也被称作黄金分割（golden section或golden mean）。

1. 计算黄金比例。

我们将使用`sqrt`函数计算5的平方根：

```
phi = (1 + numpy.sqrt(5))/2
print "Phi", phi
```

这样就得到了黄金分割数：

```
Phi 1.61803398875
```

2. 确定小于四百万的项的最大索引值。

接下来，需要确定斐波那契数列中小于四百万的项的最大索引值。我们将用维基百科页面中给出的一个公式计算这个索引值。我们需要做的是使用`log`函数，把对数的底数转换一下。不需要对计算的结果向下取整，本攻略的下一步骤将自动完成取整操作。

```
n = numpy.log(4 * 10 ** 6 * numpy.sqrt(5) + 0.5)/numpy.log(phi)
print n
```

n的计算结果是：

```
33.2629480359
```

---

① 也可以认为斐波那契数列的第一项为1，把0看作第零项并省略不写。——译者注

3. 创建一个从1到n的数组。

arange函数是一个非常基本的函数，想必大家都熟悉。为了内容的完整性，我们这里还是专门提一下。

```
n = numpy.arange(1, n)
```

4. 计算斐波那契数列。

有一个方便的公式，可以用来计算斐波那契数列。需要把黄金比例和上一步骤创建的数组作为该公式的输入参数。

为了检查计算的结果，把计算得到的斐波那契数列的前9个数打印出来：

```
fib = (phi**n - (-1/phi)**n)/numpy.sqrt(5)
print "First 9 Fibonacci Numbers", fib[:9]
```



单元测试用来测试一个小的代码单元（例如函数）的正确性。可以用单元测试代替打印语句，这个作为练习留给读者自己完成。



请参阅第8章，了解怎样进行单元测试。

顺便注意一下，数列的第一项是1。上述代码生成了我们预期的数列：

```
First 9 Fibonacci Numbers [  1.   1.   2.   3.   5.   8.  13.  21.  34.]
```

如果你愿意，可以把这个结果用在单元测试中。

5. 转换为整数。

这个步骤是可选的，但我想最终结果最好还是转换为整数。实际上，我是想介绍astype函数。

```
fib = fib.astype(int)
print "Integers", fib
```

上述代码生成如下结果（简洁起见，省略了部分内容）：

```
Integers [       1        1        2        3        5        8       13       21       34
  ... 省略 ... 省略 ...
  317811  514229  832040 1346269 2178309 3524578]
```

6. 选出数列中取值为偶数的项。

本攻略要求我们选出数列中取值为偶数的项。如果已经学习过了上一章中的布尔型索引，这

应该不难实现。

```
eventerms = fib[fib % 2 == 0]
print eventerms
```

我们得到的是：

```
[      2       8      34     144     610    2584   10946   46368   196418
  832040 3524578]
```

本攻略的完整代码如下。

```
import numpy

#斐波那契数列中的每一新项都是由该项之前的两项相加得到的。
#如果第一项和第二项分别为1和2，数列的前十项如下：

#1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

#只考虑斐波那契数列中取值不超过四百万的项，
#对其中取值为偶数的项进行求和运算。

#1. 计算黄金比例phi
phi = (1 + numpy.sqrt(5))/2
print "Phi", phi

#2. 确定取值小于四百万的项的最大索引值
n = numpy.log(4 * 10 ** 6 * numpy.sqrt(5) + 0.5)/numpy.log(phi)
print n

#3. 创建一个从1到n的数组
n = numpy.arange(1, n)
print n

#4. 计算斐波那契数列
fib = (phi**n - (-1/phi)**n)/numpy.sqrt(5)
print "First 9 Fibonacci Numbers", fib[:9]

#5. 转换为整数
# 可选的步骤
fib = fib.astype(int)
print "Integers", fib

#6. 选出取值为偶数的项
eventerms = fib[fib % 2 == 0]
print eventerms

#7. 对选出的项求和
print eventerms.sum()
```

3.2.2　攻略小结

在本攻略中，我们用到了sqrt、log、arange、astype和sum函数。它们的功能描述如下：

函　　数	功能描述
sqrt	计算数组元素的平方根
log	计算数组元素的自然对数
arange	生成一个指定范围的数组
astype	把数组元素转换为指定的数据类型
sum	计算数组元素之和

3.2.3　参考阅读

- 2.9节“布尔型索引”

3.3　寻找质因数

质因数（http://en.wikipedia.org/wiki/Prime_factor）是可以整除某个整数的质数。通过寻找质因数的方式破解密码看似几乎不可能，但如果使用正确的算法——Fermat因式分解法（http://en.wikipedia.org/wiki/Fermat%27s_factorization_method）和NumPy，这将变得很容易。基本思路是使用如下公式把整数N分解为c和d两个数：

$$N = cd = (a + b)(a - b) = a^2 - b^2$$

递归地应用这个因式分解法，直至得到需要的质因数。

3.3.1　具体步骤

该算法需要我们为上述公式中的a尝试选择若干取值。

1. 创建尝试值数组。

创建一个NumPy数组，避免使用循环语句，这是一个合理的做法。但要注意，不能创建太大的、消耗内存太多的数组。在我用的系统中，使用包含一百万个元素的数组看似是没问题的。

```
a = numpy.ceil(numpy.sqrt(n))
lim = min(n, LIM)
a = numpy.arange(a, a + lim)
b2 = a ** 2 - n
```

这里使用ceil函数对其输入参数的数组元素向上取整。

2. 得到数组b的小数部分[①]。

我们将检查数组b2中的元素是否是某个整数的平方。使用NumPy中的modf函数，可以获得数组b的小数部分。

```
fractions = numpy.modf(numpy.sqrt(b2))[0]
```

3. 查找小数部分为0的数组元素。

使用NumPy中的where函数，获得数组fractions中取值为0的元素的索引值。

```
indices = numpy.where(fractions == 0)
```

4. 找到第一个小数部分为0的数组元素。

实际上，我们只需要第一个小数部分为0的数组元素。首先调用NumPy的take函数，把上一步骤得到的数组indices作为参数，获得数组a中小数部分为0的数组元素。然后需要使用NumPy的ravel函数，把得到的结果展开为一维数组。

```
a = numpy.ravel(numpy.take(a, indices))[0]
```

如果需要解决的问题是查找数字600851475143的最大质因数，完整代码如下。

```
import numpy

#13195的质因数是5、7、13和29。

#数字600851475143的最大质因数是多少？


N = 600851475143
LIM = 10 ** 6

def factor(n):
   #1. 创建尝试值数组
   a = numpy.ceil(numpy.sqrt(n))
   lim = min(n, LIM)
   a = numpy.arange(a, a + lim)
   b2 = a ** 2 - n

   #2. 检查数组b2中的元素是否是某个整数的平方
   fractions = numpy.modf(numpy.sqrt(b2))[0]

   #3. 查找小数部分为0的数组元素
   indices = numpy.where(fractions == 0)
```

① b = numpy.sqrt(b2)。——译者注

```
    #4. 找到第一个小数部分为0的数组元素
    a = numpy.ravel(numpy.take(a, indices))[0]
    a = int(a)
    b = numpy.sqrt(a ** 2 - n)
    b = int(b)
    c = a + b
    d = a - b

    if c == 1 or d == 1:
       return

    print c, d
    factor(c)
    factor(d)

factor(N)
```

这段代码的执行结果如下。

```
1234169 486847
1471 839
6857 71
```

3.3.2　攻略小结

我们采用了递归的方式应用Fermat因式分解法，用到了NumPy中的`ceil`、`modf`、`where`、`ravel`和`take`函数。这些函数的功能描述如下。

函　数	功能描述
ceil	对数组元素向上取整
modf	返回浮点数的小数部分和整数部分
where	返回取值符合条件的数组元素的索引值
ravel	返回一个展开的一维数组
take	从数组中取出指定的元素

3.4　寻找回文数

回文数是指从左往右读和从右往左读都一样的数字。两个两位数相乘，可获得的最大回文数是$9009=91\times99$。让我们试着寻找由两个三位数相乘而获得的最大回文数。

3.4.1　具体步骤

我们将使用NumPy中的`arange`函数，创建一个包含全部三位数字（100~999）的数组。

1. 创建一个由三位数构成的数组。

用numpy.testing包中的assert_equal函数，检查数组中的第一个和最后一个元素是否正确。

```
a = numpy.arange(100, 1000)
numpy.testing.assert_equal(100, a[0])
numpy.testing.assert_equal(999, a[-1])
```

2. 创建乘积数组。

我们将创建一个数组，用来存放所有可能的三位数两两相乘的结果。可以用outer函数实现这个外积运算，再把得到的数组用ravel函数展开，以便进行查找。还需要调用数组对象的sort方法，确保数组元素已正确排序。之后，我们可以做一些正确性检查。


```
numbers = numpy.outer(a, a)
numbers = numpy.ravel(numbers)
numbers.sort()
numpy.testing.assert_equal(810000, len(numbers))
numpy.testing.assert_equal(10000, numbers[0])
numpy.testing.assert_equal(998001, numbers[-1])
```

完整代码如下。

```
import numpy
import numpy.testing

#回文数从左往右读和从右往左读都一样。
#两个两位数相乘可获得的最大回文数是9009 = 91 x 99。

#寻找由两个三位数相乘而获得的最大回文数。


#1. 创建一个由三位数构成的数组
a = numpy.arange(100, 1000)
numpy.testing.assert_equal(100, a[0])
numpy.testing.assert_equal(999, a[-1])

#2. 创建乘积数组
numbers = numpy.outer(a, a)
numbers = numpy.ravel(numbers)
numbers.sort()
numpy.testing.assert_equal(810000, len(numbers))
numpy.testing.assert_equal(10000, numbers[0])
numpy.testing.assert_equal(998001, numbers[-1])

#3. 查找最大回文数
for i in xrange(-1, -1 * len(numbers), -1):
   s = str(numbers[i])
```

```
    if s == s[::-1]:
        print s
        break
```

这段代码的执行结果是显示出906609这个回文数。

3.4.2　攻略小结

我们看到了outer函数的实际应用，该函数返回的是两个数组的外积（http://en.wikipedia.org/wiki/Outer_product）。sort函数返回的是某个数组排序后的副本。

3.4.3　更多工作

对结果进行进一步的查验是个好主意。稍微修改一下代码，弄清楚我们得到的回文数是由哪两个三位数相乘后生成的。请试着使用NumPy实现这个步骤。

3.5　确定稳态向量

马尔科夫链（Markov chain）被用来表示至少有两个状态的系统。有关马尔科夫链的详细信息，请参阅http://en.wikipedia.org/wiki/Markov_chain。此类系统t时刻的状态仅取决于t−1时刻的状态。系统的当前状态在这些状态之间随机地切换。如果把一支股票的股价变动情况定义为一个马尔科夫链，并定义持平F、上涨U和下跌D这三个状态，则我们可以根据当日收盘价确定其稳态。

在未来某个时刻之后，或者从理论上讲经过无限长时间之后，马尔科夫链系统的状态将不再改变。这个状态被称为稳态（http://en.wikipedia.org/wiki/Steady_state）。随机矩阵（http://en.wikipedia.org/wiki/Stochastic_matrix）包含了状态之间转移的概率。把随机矩阵A应用于稳态，状态将保持不变，用数学符号表示如下：

$$Ax = x$$

也可以把稳态看作特征值为1的特征向量（http://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors）。

3.5.1　具体步骤

现在我们需要先获取数据。

1. 获取一年的数据。

获取数据的一种办法是使用Matplotlib（如有必要，请参阅1.5节）。我们将获取从现在算起一年内的数据，相关代码如下。

```
today = date.today()
start = (today.year - 1, today.month, today.day)

quotes = quotes_historical_yahoo('AAPL', start, today)
```

2. 选取收盘价。

我们已经从雅虎财经频道获得了需要的历史数据。这些数据被表示为一个由元组构成的列表，但我们只关心元组中包含的收盘价。历史数据列表的例子如下。

```
[(734744.0, 675.25, 673.47000000000003, 677.66999999999996,
672.60000000000002, 7228500.0), …,(734745.0, 670.63999999999999,
663.87, 671.54999999999995, 662.85000000000002, 10799600.0)]
```

元组中的第一个元素代表日期，后面依次是开盘价、最高价、最低价和收盘价，最后一个元素是成交量。可以用如下方式选择收盘价。

```
close =  [q[4] for q in quotes]
```

收盘价是每个元组中的第五个数。现在我们获得了一个由大约253个收盘价构成的列表。

3. 确定状态数组。

使用NumPy的`diff`函数获得两个相邻交易日的收盘价差，进而确定状态数组。状态数组中的每个状态由价差的符号决定。当NumPy的`sign`函数的输入参数为负值、正值和零时，其返回结果分别为–1、1和0。

```
states = numpy.sign(numpy.diff(close))
```

4. 初始化随机矩阵。

对于每一次状态转换，其起始状态和结束状态都有三种可能。例如，如果起始状态为U，则可能切换到：

- 状态U
- 状态F
- 状态D

使用NumPy的`zeros`函数，对随机矩阵进行初始化。

```
NDIM = 3
SM = numpy.zeros((NDIM, NDIM))
```

5. 选取每一种符号对应的起始状态索引。

代码现在变得有点复杂了，我们不得不使用循环语句。对每一种可能的符号，使用NumPy的where函数选取其对应的起始状态索引。这里的k是一个平滑常数，后面将讨论其作用。

```
signs = [-1, 0, 1]
k = int(sys.argv[2])

for i in xrange(len(signs)):
   # 从特定符号对应的状态开始转换
    start_indices = numpy.where
      (states[:-1] == signs[i])[0]
```

6. 平滑处理和随机矩阵。

给定起始状态，可以对每种类型转换发生的次数进行计数。把计数结果除以该起始状态对应的总的转换次数，就能获得随机矩阵中相应的转换概率。顺便说一下，这不是最佳的做法，因为可能会造成过拟合。

在现实生活中，相邻的两个交易日的收盘价有可能相同。但对于易变的股票市场而言，这种可能性很小。对于转换次数为零的情况，一种应对方法是使用加法平滑（http://en.wikipedia.org/wiki/Additive_smoothing）。基本思路是在转换次数上加一个常数，从而避免出现概率为0的情况。计算随机矩阵中各个元素的代码如下。

```
N = len(start_indices) + k * NDIM

# 跳过总转换次数为零的情况
if N == 0:
    continue

# 获取结束状态的状态值
end_values = states[start_indices + 1]

for j in xrange(len(signs)):
    # 特定类型转换发生的次数
    occurrences = len
      (end_values[end_values == signs[j]])
    SM[i][j] = (occurrences + k)/float(N)
print SM
```

上述代码的功能是，对每一种可能的转换类型，基于转换发生的次数和加法平滑技术，计算其转换概率。

使用AAPL（苹果公司）的数据和平滑常数k=1，得到如下随机矩阵。

```
[[ 0.50925926  0.00925926  0.48148148]
  [ 0.33333333  0.33333333  0.33333333]
  [ 0.35135135  0.00675676  0.64189189]]
```

7. 特征值和特征向量。

为了能得到特征值和特征向量，我们需要使用NumPy的linalg模块和eig函数。

```
eig_out = numpy.linalg.eig(SM)
print eig_out
```

eig函数返回的是一个特征值数组和一个特征向量数组。

```
(array([ 1.        ,  0.15817566,  0.32630882]),
  array([[ 0.57735027,  0.74473695,  0.00297158],
       [ 0.57735027, -0.39841481, -0.99983179],
       [ 0.57735027, -0.53538072,  0.01809841]]))
```

8. 选择特征值1对应的特征向量。

现在我们只对特征值1对应的特征向量感兴趣。实际上，特征值可能不会精确地等于1，因此应该设立一个误差边界。可以寻找取值介于0.9和1.1之间的特征值对应的特征向量的索引，代码如下。

```
idx_vec = numpy.where
  (numpy.abs(eig_out[0] - 1) < 0.1)
print "Index eigenvalue 1", idx_vec

x = eig_out[1][:,idx_vec].flatten()
```

本攻略的完整代码如下。

```
from matplotlib.finance import quotes_historical_yahoo
from datetime import date
import numpy
import sys

if (len(sys.argv) != 3):
   print "Usage python %s SYMBOL k" % (sys.argv[0])
   print "For instance python %s AAPL 1" % (sys.argv[0])
   sys.exit()

today = date.today()
start = (today.year - 1, today.month, today.day)

quotes = quotes_historical_yahoo(sys.argv[1], start, today)
close =  [q[4] for q in quotes]

states = numpy.sign(numpy.diff(close))

NDIM = 3
SM = numpy.zeros((NDIM, NDIM))

signs = [-1, 0, 1]
k = int(sys.argv[2])

for i in xrange(len(signs)):
```

```
        # 从特定符号对应的状态开始转换
        start_indices = numpy.where(states[:-1] == signs[i])[0]

        N = len(start_indices) + k * NDIM

        # 跳过总转换次数为零的情况
        if N == 0:
           continue

        # 获取结束状态的状态值
        end_values = states[start_indices + 1]

        for j in xrange(len(signs)):
           # 特定类型转换发生的次数
           occurrences = len(end_values[end_values == signs[j]])
           SM[i][j] = (occurrences + k)/float(N)

print SM
eig_out = numpy.linalg.eig(SM)
print eig_out

idx_vec = numpy.where(numpy.abs(eig_out[0] - 1) < 0.1)
print "Index eigenvalue 1", idx_vec

x = eig_out[1][:,idx_vec].flatten()
print "Steady state vector", x
print "Check", numpy.dot(SM, x)
```

这段代码的输出结果如下所示。

```
[[ 0.4952381   0.00952381  0.4952381 ]
 [ 0.33333333  0.33333333  0.33333333]
 [ 0.34210526  0.00657895  0.65131579]]
(array([ 1.        ,  0.15328174,  0.32660547]),
  array([[ 0.57735027,  0.7424435 ,  0.00144451],
       [ 0.57735027, -0.44112882, -0.99982343],
       [ 0.57735027, -0.50416566,  0.01873551]]))
Index eigenvalue 1 (array([0]),)
Steady state vector [ 0.57735027  0.57735027  0.57735027]
Check [ 0.57735027  0.57735027  0.57735027]
```

3.5.2 攻略小结

我们没有对得到的特征向量进行归一化处理。因为我们在处理概率问题，所以特征向量中的各个元素取值之和应该为1。本例介绍了diff、sign和eig 函数，它们的功能描述如下。

函　数	功能描述
diff	计算离散差分，默认计算一阶差分
sign	返回数组元素的符号
eig	返回数组的特征值和特征向量

3.5.3 参考阅读

- 1.5节“安装Matplotlib”

3.6 发现幂律分布

为了能完成本攻略，我们现在要假设自己在运作一只对冲基金。让我们沉浸到这个假设场景中，你现在是资本市场中的一名精英了。

幂律分布存在于众多领域之中，更多相关信息参见http://en.wikipedia.org/wiki/Power_law。帕累托法则（Pareto principle）就是幂律分布的一个实例，它描述了财富分布的不均匀性。该法则告诉我们，如果按照拥有财富的多少把人们划分成组，每组人数的差异是巨大的。简单地说就是，富人不会太多，亿万富豪更少。同样，社会精英也是少数人。

使用股票的收盘价计算收益率（log returns），假设其符合幂律分布。这当然是一个大的假设，但幂律分布似乎在众多领域中广泛存在。

考虑到每笔交易涉及的交易成本，我们并不想频繁进行交易。假设我们倾向于每个月进行一次买卖操作，交易时间选择在股价出现明显修正的时候，也就是说在股价大跌时。现在的问题就是，如果我们想每隔20天左右进行一笔交易，怎样确定一个适当的交易信号。

3.6.1 具体步骤

首先需要从雅虎财经频道获取过去一年的盘后数据，然后从中提取出收盘价数据。相关的步骤已在上一篇攻略中介绍了。

1. 提取正的收益率数据。

现在要根据收盘价计算收益率，有关收益率的更多信息请参见http://en.wikipedia.org/wiki/Rate_of_return。

首先需要计算收盘价的对数，然后使用NumPy的`diff`函数，对其结果进行一阶差分运算。之后从收益率数据中，选出其中的正值。为什么要把正值选出来？这个选择其实无关紧要，关键是我想研究这些正的收益率数据。

```
logreturns = numpy.diff(numpy.log(close))
pos = logreturns[logreturns > 0]
```

2. 获得收益率的出现频率。

我们需要使用`histogram`函数，获得收益率的出现频率。该函数会进行分组计数，并返回一

个包含各组计数值的数组。[①]为了能获得一个直观的线性关系，我们需要对这些频率值取对数。

```
counts, rets = numpy.histogram(pos)
rets = rets[:-1] + (rets[1] - rets[0])/2
freqs = 1.0/(counts + 0.01)
freqs =  numpy.log(freqs)
```

3. 利用频率值和收益率数据拟合直线。

使用`ployfit`函数拟合直线。

```
p = numpy.polyfit(rets,freqs, 1)
```

4. 图示结果。

最后，使用Matplotlib绘制出结果数据和拟合后的直线。

```
matplotlib.pyplot.plot(rets, freqs, 'o')
matplotlib.pyplot.plot(rets, p[0] * rets + p[1])
matplotlib.pyplot.show()
```

我们得到了一幅漂亮的由拟合直线、收益率和频率值构成的图形。

本攻略的完整代码如下。

```
from matplotlib.finance import quotes_historical_yahoo
from datetime import date
import numpy
import sys
import matplotlib.pyplot

#1. 获取收盘价数据
today = date.today()
start = (today.year - 1, today.month, today.day)

quotes = quotes_historical_yahoo(sys.argv[1], start, today)
close =  numpy.array([q[4] for q in quotes])

#2. 提取正的收益率数据
logreturns = numpy.diff(numpy.log(close))
pos = logreturns[logreturns > 0]

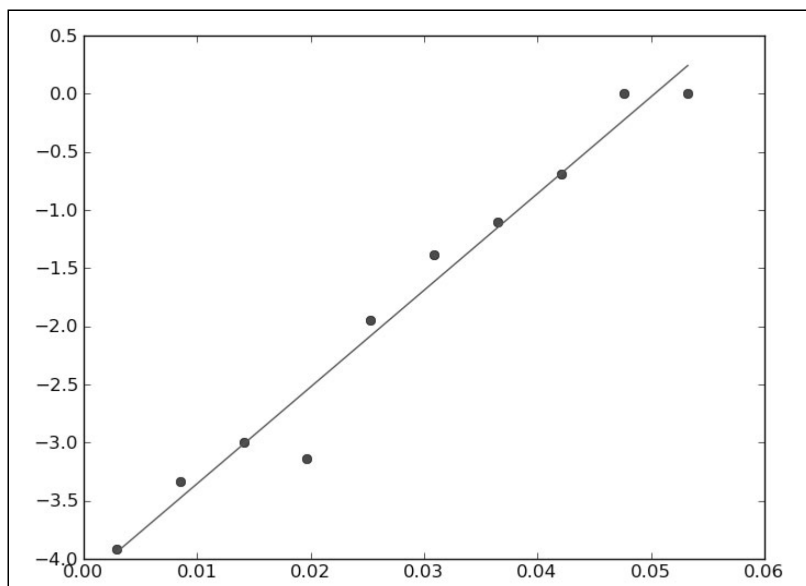
#3. 获得收益率的出现频率
counts, rets = numpy.histogram(pos)
rets = rets[:-1] + (rets[1] - rets[0])/2
freqs = 1.0/(counts + 0.01)
```

① 语句`freqs = 1.0/(counts+0.01)`中，加`0.01`目的是为了避免可能的除零错误，因为`counts`中可能会存在取值为0的计数值。——译者注

```
freqs =  numpy.log(freqs)

#4. 利用频率值和收益率数据拟合直线
p = numpy.polyfit(rets,freqs, 1)

#5. 图示结果
matplotlib.pyplot.plot(rets, freqs, 'o')
matplotlib.pyplot.plot(rets, p[0] * rets + p[1])
matplotlib.pyplot.show()
```



3.6.2　攻略小结

histgram函数利用输入的数据集计算直方图，其返回值为直方图数据和各组的界限值。polyfit函数把输入数据拟合为指定阶数的多项式曲线，本例中我们选择的是线性拟合。我们“发现”了一个幂律分布——宣布此类结论需要慎重，但直线拟合的结果让人觉得很有希望。

3.6.3　参考阅读

- 1.5节“安装Matplotlib”

3.7　定期在低点做交易

股票价格会经历周期性的下跌和上涨。我们将观察股价收益率的概率分布。

首先需要下载某只股票（例如AAPL）的历史数据，然后计算其收盘价的日收益率（http://en.wikipedia.org/wiki/Rate_of_return）。因为在之前的攻略中已详细介绍，所以这里省略了相关的步骤。

3.7.1 准备工作

如有必要，请先安装Matplotlib和SciPy。对应的攻略请见后面的“参考阅读”小节。

3.7.2 具体步骤

有趣的部分来了！

1. 计算突破和回调位置。

假设我们想每年交易五次，即大约每50天交易一次。一种策略是在股价回调的时候（下跌一定的百分比时）买入，在股价突破的时候（上涨一定的百分比时）卖出。

根据我们选定的交易频率设置适当的百分位数，换算出与之匹配的收益率。使用SciPy中的`scoreatpercentile`函数，具体实现如下。

```
freq = 1/float(sys.argv[2])
breakout = scipy.stats.scoreatpercentile
  (logreturns, 100 * (1 - freq) )
pullback = scipy.stats.scoreatpercentile
  (logreturns, 100 * freq)
```

2. 生成买入点和卖出点。

使用NumPy的`compress`函数，利用收盘价数据生成买入点和卖出点。①该函数依据给定条件确定索引位置并返回这些索引对应的数组元素。

```
buys = numpy.compress
  (logreturns < pullback, close)
sells = numpy.compress
  (logreturns > breakout, close)
print buys
print sells
print len(buys), len(sells)
print sells.sum() - buys.sum()
```

① 也许是出于简化代码的考虑，这段代码不能保证卖出点一定会晚于买入点。——译者注

如果选择AAPL（苹果公司），交易周期设定为50天，输出如下。

```
[ 340.1   377.35  378.    373.17  415.99]
[ 357.    370.8   366.48  395.2   419.55]
5 5
24.42
```

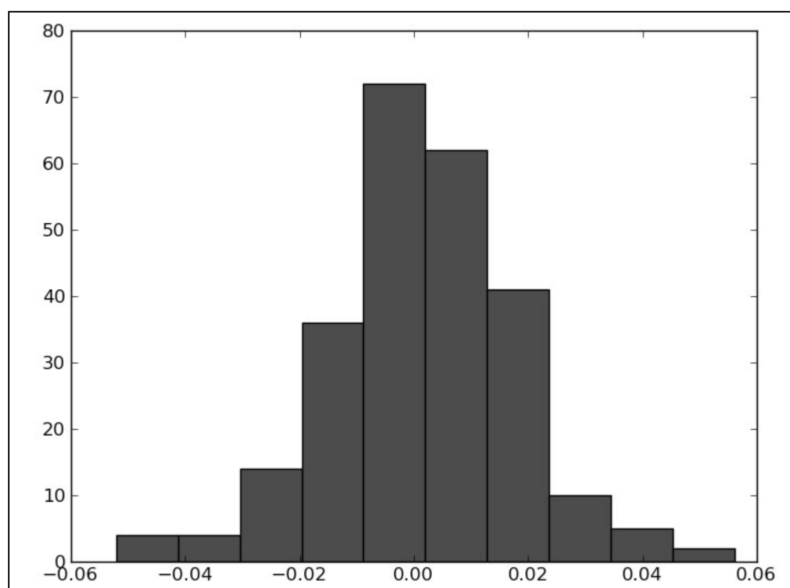
可见，如果我们买卖AAPL股票五次，将获得24美元的收益。

3. 绘制收益率直方图。

使用Matplotlib，为收益率数据绘制直方图。

```
matplotlib.pyplot.hist(logreturns)
matplotlib.pyplot.show()
```

得到如下的直方图。



本攻略的完整代码如下。

```
from matplotlib.finance import quotes_historical_yahoo
from datetime import date
import numpy
import sys
import scipy.stats
import matplotlib.pyplot

#1. 获取收盘价数据
```

```
today = date.today()
start = (today.year - 1, today.month, today.day)

quotes = quotes_historical_yahoo(sys.argv[1], start, today)
close =  numpy.array([q[4] for q in quotes])

#2. 获取收益率数据
logreturns = numpy.diff(numpy.log(close))

#3. 计算突破和回调位置
freq = 1/float(sys.argv[2])
breakout = scipy.stats.scoreatpercentile(logreturns, 100 * (1 - freq) )
pullback = scipy.stats.scoreatpercentile(logreturns, 100 * freq)

#4. 生成买入点和卖出点
buys = numpy.compress(logreturns < pullback, close)
sells = numpy.compress(logreturns > breakout, close)
print buys
print sells
print len(buys), len(sells)
print sells.sum() - buys.sum()

#5. 绘制收益率直方图
matplotlib.pyplot.hist(logreturns)
matplotlib.pyplot.show()

#AAPL 50
#[ 340.1   377.35  378.    373.17  415.99]
#[ 357.    370.8   366.48  395.2   419.55]
#5 5
#24.42
```

3.7.3 攻略小结

compress函数返回一个数组，其中包含了输入数组中满足指定条件的元素。输入数组的内容保持不变。

3.7.4 参考阅读

- 1.5节“安装Matplotlib”
- 2.2节“安装SciPy”
- 3.6节“发现幂律分布”

3.8 模拟在随机时间点做交易

上一篇攻略中，我们尝试了一种交易策略。但是因为没有测试基准，我们很难对这种交易策略的效果进行评价。这种情况下，通常会假设我们能“击败”随机过程，把在随机时间点做交易作为测试基准。我们将要模拟的是，从一年所有的交易日中，随机抽取若干日进行交易。这将展示NumPy中随机数的使用。

3.8.1 准备工作

如有必要，请先安装Matplotlib。对应的攻略请见后面的“参考阅读”小节。

3.8.2 具体步骤

首先，我们需要一个由随机整数构成的数组。

1. 生成随机索引。

使用NumPy的`randint`函数，生成随机整数。从全年交易日中随机选取若干日时，将要用到这个函数。

```
return numpy.random.randint(0, high, size)
```

2. 模拟交易过程。

模拟交易过程，需要用到上一步骤获得的随机索引。使用NumPy的`take`函数，从收盘价数组中随机选取若干元素。

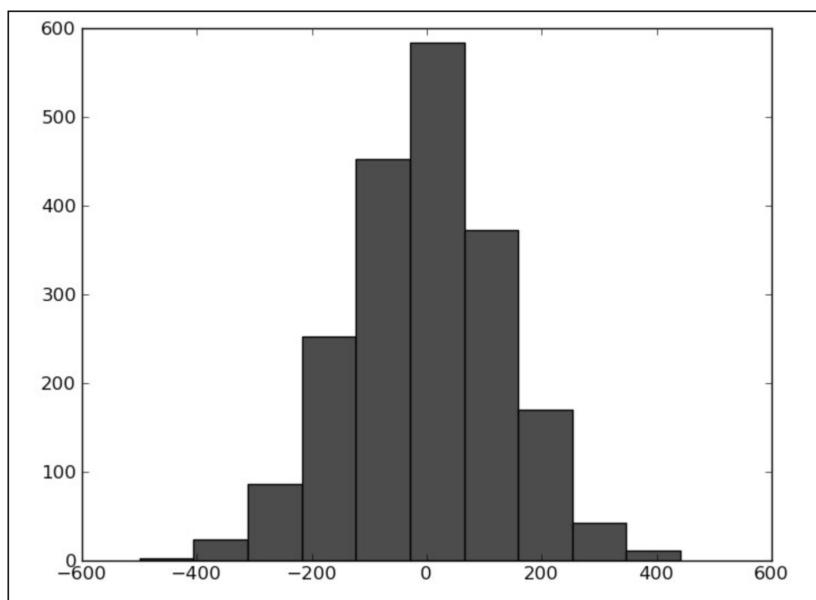
```
buys = numpy.take(close, get_indices(len(close), nbuys))
sells = numpy.take(close, get_indices(len(close), nbuys))
profits[i] = sells.sum() - buys.sum()
```

3. 绘制利润直方图。

进行很多次模拟后，绘制利润直方图。

```
matplotlib.pyplot.hist(profits)
matplotlib.pyplot.show()
```

假设使用AAPL股票的交易数据，一年买卖五次，进行2000次模拟后，得到如下的直方图。



本攻略的完整代码如下。

```
from matplotlib.finance import quotes_historical_yahoo
from datetime import date
import numpy
import sys
import matplotlib.pyplot

def get_indices(high, size):
    #2. 生成随机索引
    return numpy.random.randint(0, high, size)

#1. 获取收盘价
today = date.today()
start = (today.year - 1, today.month, today.day)

quotes = quotes_historical_yahoo(sys.argv[1], start, today)
close =  numpy.array([q[4] for q in quotes])

nbuys = int(sys.argv[2])
N = int(sys.argv[3])
profits = numpy.zeros(N)

for i in xrange(N):
   #3. 模拟交易过程
   buys = numpy.take(close, get_indices(len(close), nbuys))
   sells = numpy.take(close, get_indices(len(close), nbuys))
   profits[i] = sells.sum() - buys.sum()
```

```
print "Mean", profits.mean()
print "Std", profits.std()

#4. 绘制利润直方图
matplotlib.pyplot.hist(profits)
matplotlib.pyplot.show()

#python random_periodic.py AAPL 5 2000
#Mean -2.566465
#Std 133.746039463
```


3.8.3　攻略小结

我们用到了numpy.random模块中的`randint`函数。numpy.random模块中还包含了更多方便易用的随机数生成器，这些函数的功能描述如下。

函　数	功能描述
`rand`	根据维度参数生成指定形状的数组。数组元素是一个随机数，符合[0,1]区间上的均匀分布。如果没有指定维度参数，则返回一个浮点数
`randn`	数组元素是来自一个均值为0、方差为1的正态分布的采样值。维度参数的作用同rand
`randint`	给定一个下边界、一个可选的上边界和一个可选的输出形状，返回一个整数数组

3.8.4　参考阅读

- ❑ 1.5节“安装Matplotlib”

3.9　用埃氏筛筛选整数

埃拉托斯特尼筛法（http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes）简称埃氏筛，是一个筛选质数的算法。该算法用迭代的方式识别出已经找到的质数的倍数，能高效地筛选小于一千万的质数。让我们试着去寻找第10 001个质数。

具体步骤

首先必须要做的事情，就是创建一个自然数列表。

1. 创建一个连续的整数列表。

使用NumPy中的`arange`函数创建数组。

```
a = numpy.arange(i, i + LIM, 2)
```

2. 筛选出p的倍数。

不清楚埃拉托斯特尼本人是否希望我们这样实现算法，但确实可以这样做。把数组中能被p整除的元素移除，代码如下。

```
a = a[a % p != 0]
```

本攻略的完整代码如下。

```
import numpy

LIM = 10 ** 6
N = 10 ** 9
P = 10001
primes = []
p = 2

#通过列出前6个质数：2、3、5、7、11和13，我们看到第6个质数是13。
#第10 001个质数是多少？

def check_primes(a, p):
   #2. 筛选出p的倍数
   a = a[a % p != 0]

   return a
for i in xrange(3, N, LIM):
   #1. 创建一个连续的整数列表
   a = numpy.arange(i, i + LIM, 2)

   while len(primes) < P:
      a = check_primes(a, p)
      primes.append(p)

      p = a[0]

print len(primes), primes[P-1]
```

第 4 章

NumPy与其他软件的交互

本章主要内容：

- 使用缓冲区协议
- 使用数组接口
- 与MATLAB和Octave交换数据
- 安装RPy2
- 连接到R
- 安装JPype
- 传递NumPy数组到JPype
- 安装谷歌应用程序引擎
- 在谷歌云中部署NumPy代码
- 在Python Anywhere的Web控制台中运行NumPy代码
- 设置PiCloud

4.1 引言

本章的主题是*互操作性*。我们需要不断地提醒自己，在Python科学计算软件这个生态系统中，NumPy不是孤立的。NumPy与SciPy和Matplotlib协同工作是相当容易的，与其他Python软件包进行互操作的协议也存在。在Python生态系统之外，诸如Java、R、C和Fortran等编程语言也非常流行。我们将会讨论与这些编程环境交换数据的细节。

此外，我们将讨论怎样让NumPy代码运行在云端。这是一个正在持续快速演进的技术。在实现技术方面有很多选择，我们将介绍谷歌应用程序引擎、PiCloud和Python Anywhere。



选择介绍哪家公司的产品存在一定的风险，因为有可能被认为掺杂了主观因素。请相信本书作者和上述的所有公司绝对没有利益联系。

4.2　使用缓冲区协议

用C语言实现的Python对象有一个所谓的“缓冲区接口”。无需复制，此类Python对象允许其他对象直接访问它们的数据成员。缓冲区协议使我们能方便地在NumPy和其他Python软件（例如Python图像库PIL）之间交换信息。我们来看一个直接把NumPy数组保存为PIL图像的例子。

4.2.1　准备工作

如有必要，请先安装PIL和SciPy。对应的攻略请见后面的“参考阅读”小节。

4.2.2　具体步骤

首先，我们需要一个操作对象——NumPy数组。

1. 用图像数据创建一个数组。

在之前的章节中，我们已经介绍了怎样加载范例图像Lena。我们将创建一个用0填充的数组，并且把图像数据填充到alpha通道中。

```
lena = scipy.misc.lena()
data = numpy.zeros((lena.shape[0], lena.shape[1], 4), dtype=numpy.int8)
data[:,:,3] = lena.copy()
```

2. 把数据保存为PIL图像。

我们现在使用PIL API，把数据保存为RGBA格式的图像。

```
img = Image.frombuffer("RGBA", lena.shape, data)
img.save('lena_frombuffer.png')
```

3. 修改数组`data`并保存图像。

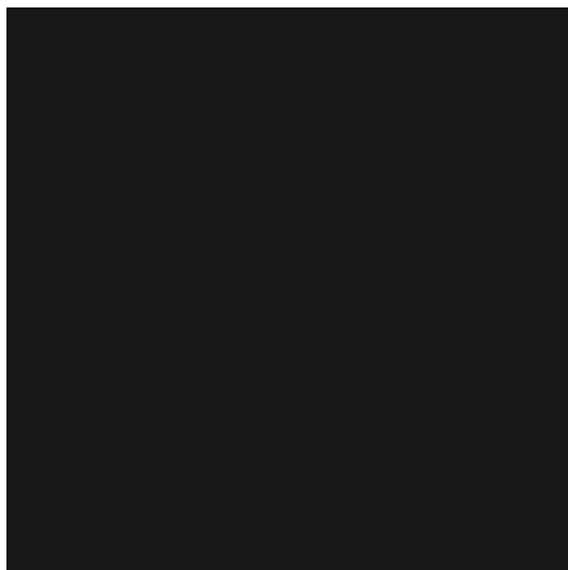
修改数组`data`，删除图像数据，使图像显示为红色。用PIL API保存图像。

```
data[:,:,3] = 255
data[:,:,0] = 222
img.save('lena_modified.png')
```

下图是修改数组内容之前保存的图像。



由于使用了缓冲区接口，数组data的改变会导致PIL图像对象的数据改变。因此，我们会看到如下图像。



本攻略的完整代码如下。

```
import numpy
import Image
import scipy.misc
```

```
lena = scipy.misc.lena()
data = numpy.zeros((lena.shape[0], lena.shape[1], 4), dtype=numpy.int8)
data[:,:,3] = lena.copy()
img = Image.frombuffer("RGBA", lena.shape, data)
img.save('lena_frombuffer.png')

data[:,:,3] = 255
data[:,:,0] = 222
img.save('lena_modified.png')
```

4.2.3　攻略小结

我们从一个缓冲区——NumPy数组，创建了一个PIL图像。改变缓冲区内容后，我们看到图像对象也会相应发生改变。实现这些并不需要复制PIL图像对象。我们直接访问和修改NumPy数组，把一个模特的肖像变成了一个红色图像。

4.2.4　参考阅读

- 2.3节“安装PIL”
- 2.2节“安装SciPy”

4.3　使用数组接口

数组接口是另一种与其他Python应用程序通信的机制。顾名思义，该协议机制只适用于类数组（array-like）对象。接下来还是使用PIL举例，但不涉及保存文件的操作。

4.3.1　准备工作

我们将复用上一攻略的部分代码，因此相关的准备工作也是类似的。这里省略了对上一攻略中的第一个步骤的介绍，假定你已知道怎样用图像数据创建数组。

4.3.2　具体步骤

让我们开始探索数组接口的用法。

1. PIL图像的数组接口属性。

PIL图像对象有一个`__array_interface__`属性。属性值是一个字典对象。让我们看一下它的内容。

```
array_interface = img.__array_interface__
print "Keys", array_interface.keys()
print "Shape", array_interface['shape']
print "Typestr", array_interface['typestr']
```

这段代码会打印如下信息。

```
Keys ['shape', 'data', 'typestr']
Shape (512, 512, 4)
Typestr |u1
```

2. NumPy数组接口属性。

NumPy的ndarray模块也有一个`__array_interface__`属性。可以使用`asarray`函数把PIL图像转换为NumPy数组。

```
numpy_array = numpy.asarray(img)
print "Shape", numpy_array.shape
print "Data type", numpy_array.dtype
```

该数组的形状和数据类型是：

```
Shape (512, 512, 4) Data type uint8
```

如你所见，数组的形状没有改变。本攻略的完整代码如下。

```
import numpy
import Image
import scipy.misc

lena = scipy.misc.lena()
data = numpy.zeros((lena.shape[0], lena.shape[1], 4), dtype=numpy.int8)
data[:,:,3] = lena.copy()
img = Image.frombuffer("RGBA", lena.shape, data)
array_interface = img.__array_interface__
print "Keys", array_interface.keys()
print "Shape", array_interface['shape']
print "Typestr", array_interface['typestr']

numpy_array = numpy.asarray(img)
print "Shape", numpy_array.shape
print "Data type", numpy_array.dtype
```

4.3.3 攻略小结

数组接口（协议）使我们能够在Python的类数组对象之间共享数据。NumPy和PIL都提供了数组接口。

4.3.4　参考阅读

- 4.2节“使用缓冲区协议”

4.4　与 MATLAB 和 Octave 交换数据

MATLAB和它的开源替代品Octave都是流行的数学软件。scipy.io包中有一个savemat函数。用NumPy数组构造一个字典对象。savemat函数可以把该字典对象转换并存储到一个.mat文件中。

4.4.1　准备工作

怎样安装MATLAB和Octave不在本书的讨论范围之内。Octave的官网上有安装指南（http://www.gnu.org/software/octave/download.html）。如果需要了解怎样安装SciPy，请先看后面的“参考阅读”小节。

4.4.2　具体步骤

安装好MATLAB或Octave后，需要按照如下步骤保存NumPy数组。

1. 调用savemat函数。

创建一个NumPy数组。调用savemat函数，把这个数组保存到一个.mat文件中。savemat函数需要两个参数：文件名，包含变量名和变量取值的字典对象。

```
a = numpy.arange(7)
scipy.io.savemat("a.mat", {"array": a})
```

2. 加载.mat文件。

切换到上一步骤生成的.mat文件所在的目录。在Octave中加载该文件，并查验数组数据。

```
octave-3.4.0:2> load a.mat
octave-3.4.0:3> array
array =

      0
      1
      2
      3
      4
      5
      6
```

本攻略的完整代码如下。

```
import numpy
import scipy.io

a = numpy.arange(7)
scipy.io.savemat("a.mat", {"array": a})
```

4.4.3　参考阅读

❑ 2.2节“安装SciPy”

4.5　安装 RPy2

R是一种流行的用来做统计和数据分析的脚本语言。RPy2是R和Python的接口。本攻略将介绍RPy2的安装方法。

具体步骤

安装RPy2的方法如下，选择其一即可。

▶ 使用pip或easy_install安装

在PYPI上有RPy2包，因此可以使用如下两条命令之一完成安装。

```
easy_install rpy2
```

或者

```
sudo pip install rpy2
```

▶ 从源文件安装

用tar.gz格式的源文件安装RPy2。

```
tar -xzf <rpy2_package>.tar.gz
cd <rpy2_package>
python setup.py build install
```

4.6　连接到 R

RPy2只能用来从Python调用R，不支持反向的调用。我们将导入R中的一些范例数据集，并把其中的一组数据用图形化的方式表示。

4.6.1　准备工作

如有必要，请先安装RPy2。参见上一篇攻略。

4.6.2　具体步骤

首先要加载R中的一个范例数据集。

1. 把一个数据集加载到数组中。

使用RPy2的`importr`函数加载数据集。该函数用来导入R包。本例中，我们将导入R中的datasets包。用mtcars数据集创建一个NumPy数组。

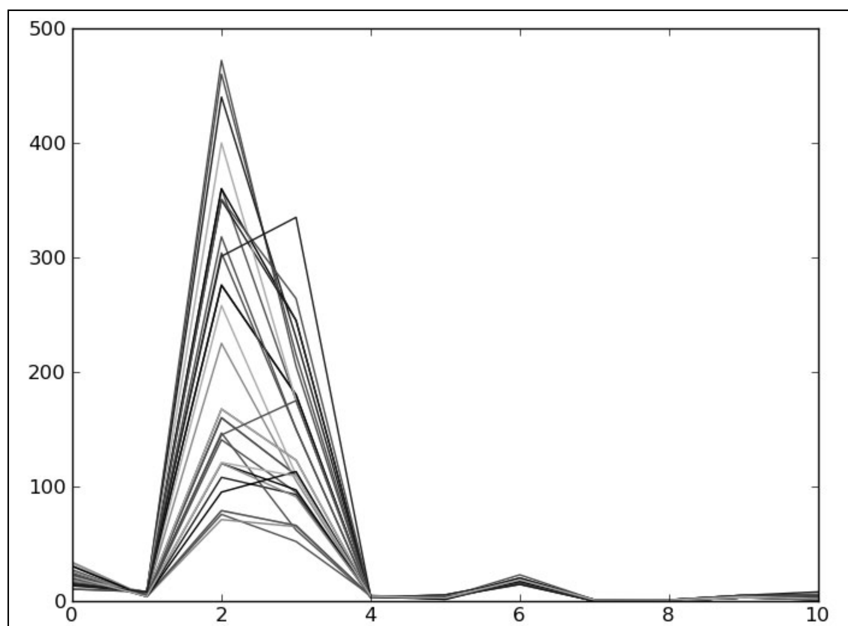
```
datasets = importr('datasets')
mtcars = numpy.array(datasets.mtcars)
```

2. 绘制数据集。

使用Matplotlib，绘制出mtcars数据集。

```
matplotlib.pyplot.plot(mtcars)
matplotlib.pyplot.show()
```

该数据集被表示为一个二维数组。绘制的结果如下图所示。



本攻略的完整代码如下。

```
from rpy2.robjects.packages import importr
import numpy
import matplotlib.pyplot

datasets = importr('datasets')
mtcars = numpy.array(datasets.mtcars)

matplotlib.pyplot.plot(mtcars)
matplotlib.pyplot.show()
```

4.6.3 参考阅读

- 1.5节“安装Matplotlib”

4.7 安装 JPype

Jython是实现Python和Java之间互操作性的默认解决方案。但Jython运行在Java虚拟机环境中，因此不能访问主要用C语言编写的NumPy模块。JPype是试图解决这个互操作问题的一个开源项目。JPype在本机级别提供了一个Python和Java虚拟机的接口。让我们开始安装JPype。

具体步骤

按照如下步骤安装JPype。

1. 下载JPype。

从http://sourceforge.net/projects/jpype/files/下载JPype的源文件。

2. 构建JPype。

解压缩JPype的源文件，并运行如下命令。

```
python setup.py install
```

4.8 传递 NumPy 数组到 JPype

在本攻略中，我们将启动一个Java虚拟机（JVM），并把一个NumPy数组发送到JVM。我们将使用标准的Java调用语法把接收到的数组打印出来。当然，你需要提前安装好Java。

4.8.1 具体步骤

首先，我们需要在JPype中启动JVM。

1. 启动JVM。

JPype可以方便地找到默认的JVM路径。

```
jpype.startJVM(jpype.getDefaultJVMPath())
```

2. 打印hello world。

出于对传统的尊重，先打印hello world。

```
jpype.java.lang.System.out.println("hello world")
```

3. 传递NumPy数组。

创建一个NumPy数组，把它转换为Python列表并传递给JPype。之后就可以轻松打印出各个数组元素。

```
values = numpy.arange(7)
java_array = jpype.JArray
  (jpype.JDouble, 1)(values.tolist())

for item in java_array:
    jpype.java.lang.System.out.println(item)
```

4. 关闭JVM。

所有的事情都完成后，关闭JVM。

```
jpype.shutdownJVM()
```

在JPype中，任何时刻都只能有一个JVM正在运行。如果忘记关闭JVM，有可能导致意想不到的错误。程序的输出如下。

```
hello world
    0.0
    1.0
    2.0
    3.0
    4.0
    5.0
    6.0
    JVM activity report     :
      classes loaded       : 31
    JVM has been shutdown
```

本攻略的完整代码如下。

```
import jpype
import numpy

#1. 启动JVM
jpype.startJVM(jpype.getDefaultJVMPath())

#2. 打印hello world
jpype.java.lang.System.out.println("hello world")

#3. 传递NumPy数组
values = numpy.arange(7)
java_array = jpype.JArray(jpype.JDouble, 1)(values.tolist())

for item in java_array:
   jpype.java.lang.System.out.println(item)

#4. 关闭JVM
jpype.shutdownJVM()
```


4.8.2　攻略小结

JPype允许我们启动和停止一个Java虚拟机。JPype提供了对标准Java API调用的封装。本例中，我们通过使用`JArray`封装，可以把Python列表传递到JVM并转换为Java数组。JPype使用了Java本地接口（JNI）。JNI实现了本地原生C语言代码和Java之间的桥接。遗憾的是，使用JNI会导致软件性能降低，因此使用JPype的时候要考虑到这一点。

4.8.3　参考阅读

- 4.7节“安装JPype”

4.9　安装谷歌应用程序引擎

谷歌应用程序引擎（GAE）使你能在谷歌云（Google cloud）中构建Web应用程序。从2012年开始，GAE正式支持NumPy。为了能使用GAE，你需要注册一个谷歌账户。

具体步骤

第一步是下载GAE。

1. 下载GAE。

从https://developers.google.com/appengine/downloads下载适合你的操作系统的GAE版本。

从该页面也能同时下载相关文档和GAE的Eclipse插件。如果使用Eclipse做开发，你绝对应该安装这个插件。

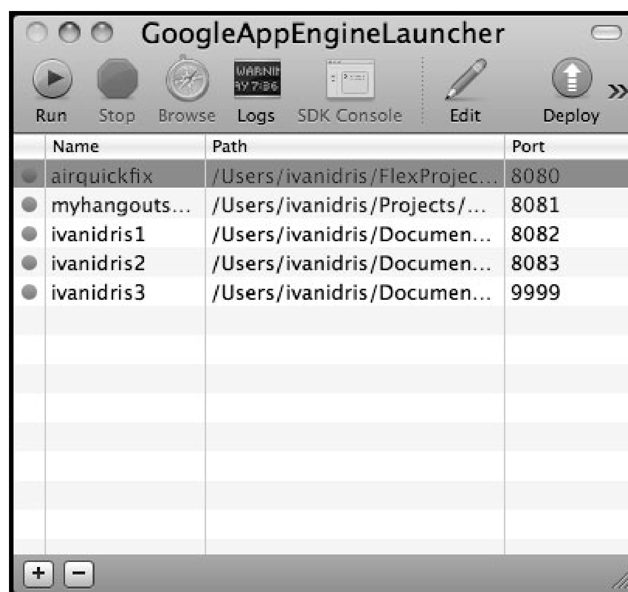
2. 开发环境。

GAE自带一个开发环境，用来在本地模拟云端产品。在写作本书的时候，GAE仅提供对Python 2.5和2.7的正式支持。GAE会试图找到系统中的Python，但有的时候，例如系统中安装了多个版本的Python，也可能需要你手动设置。可以在GAE启动器的Preferences对话框中进行相关设置。

GAE的SDK中有两个重要的脚本。

- dev_appserver.py：开发服务器
- appcfg.py：部署到云端

在Windows和Mac系统中，可以看到一个名为GAE启动器的应用程序。这个程序中的Run和Deploy按钮的功能与上述的两个脚本对应。



4.10　在谷歌云中部署 NumPy 代码

部署GAE应用是相当简单的。如果用到了NumPy，则需要一个额外的配置步骤，但也就是几分钟即可解决的事情。

4.10.1 具体步骤

让我们创建一个新的应用。

1. 创建一个新的应用。

使用GAE启动器程序，创建一个新的应用（File | New Application），并把它命名为numpycloud。这将生成一个同名的文件夹，其中包括如下内容。

- app.yaml：YAML应用程序配置文件
- favicon.ico：图标
- index.yaml：自动生成的文件
- main.py：Web应用程序的主入口

2. 把NumPy加到库中。

需要让GAE知道我们想使用NumPy。添加如下内容到配置文件app.yaml的libraries区段。

```
- name: numpy
  version: "1.6.1"
```

该配置文件应该包括如下的内容。

```
application: numpycloud
version: 1
runtime: python27
api_version: 1
threadsafe: yes

handlers:
- url: /favicon\.ico
  static_files: favicon.ico
  upload: favicon\.ico

- url: .*
  script: main.app

libraries:
- name: webapp2
  version: "2.5.1"
- name: numpy
  version: "1.6.1"
```

3. 编写NumPy代码。

为了演示NumPy代码的使用，我们修改一下main.py文件。该文件中有一个`MainHandler`类，其中包含了一个用来处理`get`请求的方法。使用如下代码替换该方法的内容.

```
def get(self):
    self.response.out.write
      ('Hello world!<br/>')
    self.response.out.write
      ('NumPy sum = ' + str
      (numpy.arange(7).sum()))
```

最终将得到如下的代码。

```
import webapp2
import numpy

class MainHandler
  (webapp2.RequestHandler):
    def get(self):
      self.response.out.write('Hello world!<br/>')
      self.response.out.write('NumPy sum = ' +
        str(numpy.arange(7).sum()))

app = webapp2.WSGIApplication([('/', MainHandler)],
                              debug=True)
```

在GAE启动器中点击Browse按钮，将在默认浏览器中看到包含如下文字的页面。

```
Hello world!
NumPy sum = 21
```

4.10.2　攻略小结

如果没有使用太多资源，GAE的使用是免费的。你最多可以创建10个Web应用。GAE采用了沙箱机制，这是过去的一段时间内不能使用NumPy的原因。但正如本攻略所示，GAE现在已经支持NumPy了。

还需要知道的是，GAE目前不支持关系数据库。GAE的一些其他特性也会影响应用的可移植性。

4.11　在 Python Anywhere 的 Web 控制台中运行 NumPy 代码

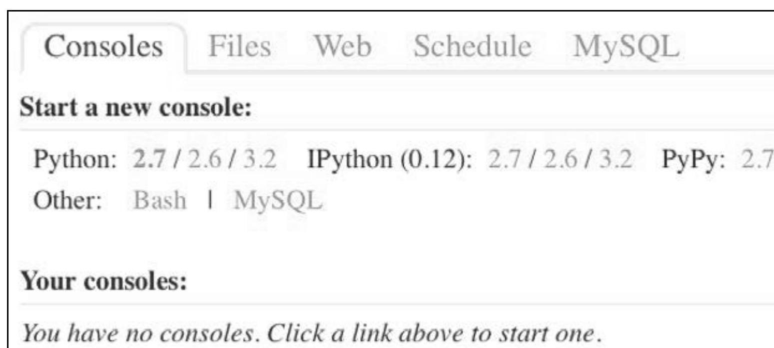
在第1章中，在没有账号的情况下，我们已经实际查看了Python Anywhere的控制台。本攻略需要你有一个账号。但不必担心，如果不需要太多资源的话，可以申请免费账号。

注册账号是一个很简单的过程，这里就不介绍了。包括NumPy在内的众多Python软件已经安装好了，这些软件的完整列表请见 https://www.pythonanywhere.com/batteries_included/。

我们将编写一个简单的脚本，用来从谷歌财经频道获取价格数据（每分钟获取一次），并用NumPy对这些价格数据做简单的统计分析。

4.11.1　具体步骤

注册账号并登录，将看到如下的Python Anywhere仪表盘（dashboard）。



1. 编写代码。

本攻略的完整代码如下。

```
import urllib2
import re
import time
import sys
import numpy

prices = numpy.array([])

for i in xrange(3):
   req = urllib2.Request
     ('http://finance.google.com/finance/
     info?client=ig&q=' + sys.argv[1])
   req.add_header('User-agent', 'Mozilla/5.0')
   response = urllib2.urlopen(req)
   page = response.read()
   m = re.search('l_cur" : "(.*)"', page)
   prices = numpy.append(prices, float(m.group(1)))
   avg = prices.mean()
   sigma = prices.std()

   devFactor = float(sys.argv[2])
   bottom = avg - devFactor * sigma
   top = avg + devFactor * sigma
   timestr = time.strftime("%H:%M:%S", time.gmtime())

   print timestr, "Average", avg,
     "-Std", bottom, "+Std", top
   time.sleep(60)
```

我们创建了一个包含股价数据的NumPy数组，并计算了这些数据的均值和标准差。除此之外都是标准的Python代码。一个URL被用来从谷歌财经频道下载JSON格式的股价数据。需要你提供股票代码，例如AAPL。当然，这个URL未来有可能改变。

接下来需要用正则表达式解析JSON格式的数据，获取股价信息并添加到NumPy数组。然后，利用这些股价数据计算均值和标准差。最后打印输出当前时间、股价均值和置信区间的上下边界。上下边界值的计算方法是用标准差乘以一个我们提供的系数。

2. 上传代码。

在本机编写完脚本代码后，就可以把它上传到Python Anywhere。在仪表盘页面中点击Files标签页，利用页面底端的插件上传该脚本。

3. 运行代码。

点击Consoles标签页，再点击Bash链接，Python Anywhere会立刻为我们创建一个bash控制台。

现在可以运行程序。如果选择AAPL股票，置信区间设定为一倍的标准差，则运行结果如下图所示。

```
18:32 ~/cookbook $ python avg_price.py AAPL 1
15:31:29 Average 667.24 -Std 667.24 +Std 667.24
15:32:29 Average 667.365 -Std 667.24 +Std 667.49
15:33:29 Average 667.376666667 -Std 667.273279584 +Std 667.480053749
```

4.11.2　攻略小结

如果你想在一台远程的服务器上运行NumPy代码，特别是想让程序在预先设定的时间做某件事情，Python Anywhere是一个很好的选择。另一方面，至少对于免费账户而言，因为在Web控制台上输入文本有一定的时延，所以在Python Anywhere上做交互性的工作并不太方便。

然而，正如我们所见，我们可以在本地创建和测试程序，然后再把它上传到Python Anywhere。这样做也节省了本地资源。我们可以做一些很酷的事情，例如根据股价高低发送电子邮件，或者调度我们的脚本在交易时间运行。顺便说一下，用谷歌应用程序引擎做此类事情也是可能的，但你需要遵循谷歌规定的方式并学习相关的API。

4.12　设置 PiCloud

PiCloud是另一个云计算服务的提供者，它实际上是基于亚马逊的EC2基础平台的。PiCloud为用户提供预安装了Python软件（包括NumPy）的环境。这些环境其实就是可以远程登录的EC2

实例。在本攻略中，我们将使用基于Ubuntu Natty 11.04和Python 2.7的环境。该环境中已经安装的软件包列表请见http://www.picloud.com/docs/base_environment/2/installed/。PiCloud遵循免费增值模式，即在开始阶段可以免费使用，以后如果需要较多资源需要付费。

4.12.1 具体步骤

注册账号并登录PiCloud。

1. 创建一个环境。

初始状态时是没有环境的。为了创建一个环境，首先点击Environment标签页，接着再点击create a new environment按钮。当前有建立在Ubuntu上的基于Python 2.7或2.6的环境可供选择，请选择Python 2.7环境。

创建环境需要几分钟时间。环境准备就绪后，你会收到一封电子邮件，并在浏览器中看到类似下图所示的信息。

Environments being configured

name	description	version	status	action
py27UbuntyNatty1104		Python 2.7 - Ubuntu Natty 11.04	ec2-107-20-19-178.compute-1.amazonaws.com In configurable mode for 00:46:04	connect save

create new environment

2. 连接到环境。

点击action栏中的connect链接，将得到一个弹出窗口，窗口中显示如下指令。

```
chmod 400 privatekey.pem
ssh -i privatekey.pem picloud@yourserver.amazonaws.com
```

从弹出窗口中给出的链接地址下载私钥，然后使用上述指令连接到环境。

3. 检查版本号。

为了证明我们可以使用NumPy和Matplotlib，需要检查一下它们的版本号。

首先启动一个IPython shell。回想一下第1章中的内容，使用pylab选项开关可以自动导入包括NumPy在内的若干个包。执行如下命令。

```
ipython -pylab
```

NumPy和Matplotlib有一个__version__属性，其内容就是版本号。打印如下版本号。

```
In [1]: numpy.__version__
Out[1]: '1.6.1'

In [4]: matplotlib.__version__
Out[4]: '1.0.1'
```

4.12.2　攻略小结

PiCloud提供了预配置的亚马逊EC2实例，其中包括了NumPy在内的各种Python软件包。可以从终端程序直接访问这些预配置的环境。可以对环境进行定制并保存定制结果，以便将来使用。定制后的环境也可以用作模版。

第 5 章

声音和图像处理

本章将介绍怎样用NumPy和SciPy进行基本的图像和声音（WAV文件）处理。我们将使用NumPy，用声音和图像做一些有趣的事情：

- 加载图像到内存映射区
- 合并图像
- 图像的模糊化处理
- 复制声音片段
- 合成声音
- 设计音频滤波器
- 用索贝尔滤波器进行边缘检测

5.1 引言

这应该是有趣的一章。虽然本书的所有章节都很有趣，但在本章中，我们将尽可能地追求乐趣，专注于做有趣的事情。在第10章中，你将看到更多使用scikits-image实现图像处理的攻略。

很遗憾本书不能直接支持声音文件的播放。你需要亲自运行范例代码，以便充分体会和了解攻略的内容。从本书官网http://www.packtpub.com/可以获得相关的源代码。

5.2 加载图像到内存映射区

建议把大文件加载到内存映射区。内存映射文件只加载大文件的一小部分。NumPy的内存映射是类数组结构。本例中，我们将生成一幅由若干彩色小方块构成的图像并将其加载到内存映射区。

5.2.1　准备工作

如有必要，请先安装Matplotlib。对应的攻略请参见后面的“参考阅读”小节。

5.2.2　具体步骤

我们将从初始化数组开始介绍。

1. 初始化数组。

首先对如下的数组进行初始化。

- 用于保存图像数据的数组
- 用于存放随机生成的小方块中心点坐标的数组
- 用于存放随机生成的小方块半边长的数组
- 用于存放随机生成的小方块颜色的数组

数组的初始化代码如下。

```
img = numpy.zeros((N, N), numpy.uint8)
centers = numpy.random.random_integers
  (0, N, size=(NSQUARES, 2))
radii = numpy.random.randint
  (0, N/9, size=NSQUARES)
colors = numpy.random.randint
  (100, 255, size=NSQUARES)
```

如你所见，第一个数组被初始化为全零数组，其他数组用numpy.random包中生成随机整数的函数进行初始化。

2. 生成小方块。

下个步骤是生成小方块。使用上一步骤创建的数组生成小方块。通过使用`clip`函数，可以确保这些小方块不会超出图像区域。

`meshgrid`函数用来生成小方块的坐标。如果该函数的输入参数是大小为N和M的两个一维数组，则会返回两个N × M的二维数组。第一个数组中的元素沿x轴方向有规律地重复，第二个数组中的元素沿y轴方向有规律地重复。下面这段IPython会话清楚地展示了`meshgrid`函数的功能。

```
In: x = linspace(1, 3, 3)

In: x
Out: array([ 1.,  2.,  3.])
```

```
In: y = linspace(1, 2, 2)

In: y
Out: array([ 1.,  2.])

In: meshgrid(x, y)
Out:
[array([[ 1.,  2.,  3.],
      [ 1.,  2.,  3.]]),
 array([[ 1.,  1.,  1.],
      [ 2.,  2.,  2.]])]
```

之后还需要设置小方块的颜色。

```
for i in xrange(NSQUARES):
    xindices = range(centers[i][0] - radii[i], centers[i][0] +
radii[i])
    xindices = numpy.clip(xindices, 0, N - 1)
    yindices = range(centers[i][1] - radii[i], centers[i][1] +
radii[i])
    yindices = numpy.clip(yindices, 0, N - 1)

    if len(xindices) == 0 or len(yindices) == 0:
      continue

    coordinates = numpy.meshgrid(xindices, yindices)
    img[coordinates] = colors[i]
```

3. 加载图像数据到内存映射区。

在加载图像数据到内存映射区之前，需要先用`tofile`函数把图像保存到文件，然后再用`memmap`函数把图像数据从文件加载到内存映射区。

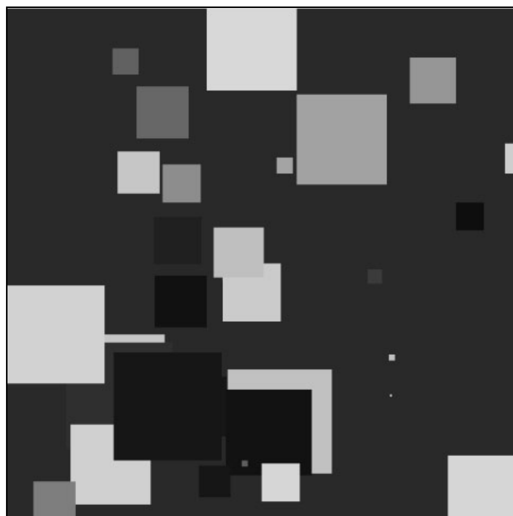
```
img.tofile('random_squares.raw')
img_memmap = numpy.memmap
  ('random_squares.raw', shape=img.shape)
```

4. 显示图像。

为了证明一切都符合预期，我们用Matplotlib显示图像。

```
matplotlib.pyplot.imshow(img_memmap)
matplotlib.pyplot.axis('off')
matplotlib.pyplot.show()
```

注意，这里设置了不显示坐标轴。最终生成的图像的一个实例如下。



本攻略的完整代码如下。

```
import numpy
import matplotlib.pyplot
import sys

N = 512

if(len(sys.argv) != 2):
   print "Please input the number of squares to generate"
   sys.exit()

NSQUARES = int(sys.argv[1])

# 初始化
img = numpy.zeros((N, N), numpy.uint8)
centers = numpy.random.random_integers(0, N, size=(NSQUARES, 2))
radii = numpy.random.randint(0, N/9, size=NSQUARES)
colors = numpy.random.randint(100, 255, size=NSQUARES)

# 生成小方块
for i in xrange(NSQUARES):
   xindices = range(centers[i][0] - radii[i], centers[i][0] + radii[i])
   xindices = numpy.clip(xindices, 0, N - 1)
   yindices = range(centers[i][1] - radii[i], centers[i][1] + radii[i])
   yindices = numpy.clip(yindices, 0, N - 1)

   if len(xindices) == 0 or len(yindices) == 0:
      continue

   coordinates = numpy.meshgrid(xindices, yindices)
```

```
    img[coordinates] = colors[i]

# 加载到内存映射区
img.tofile('random_squares.raw')
img_memmap = numpy.memmap('random_squares.raw', shape=img.shape)

# 显示图像
matplotlib.pyplot.imshow(img_memmap)
matplotlib.pyplot.axis('off')
matplotlib.pyplot.show()
```

5.2.3　攻略小结

在本攻略中，我们用到了下列函数。

函　　数	功能描述
zeros	创建一个全零数组
random_integers	返回一个整数数组，其取值为上下边界值之间的随机整数
randint	功能同random_integers
clip	给定最大值和最小值，对超出这个范围的数组元素的取值进行裁剪（使其等于最大值或最小值）
meshgrid	输入一个x坐标数组和一个y坐标数组，返回二维坐标数组
tofile	把数组内容写入文件
memmap	给定文件名，从该文件创建一个NumPy内存映射文件。可以指定数组的形状（可选）
axis	绘图时配置轴的Matplotlib函数，例如可以让轴不显示

5.2.4　参考阅读

- 1.5节“安装Matplotlib”

5.3　合并图像

在本攻略中，我们将把著名的曼德勃罗分形图（Mandelbrot fractal）和图像Lena进行合并。有关曼德勃罗集的更多信息请见http://en.wikipedia.org/wiki/Mandelbrot_set。此类分形可以被定义为一个迭代公式，当前的复数值的平方，加上一个常数，就得到下一个复数值。

5.3.1　准备工作

如有必要，请先安装SciPy。相关的攻略请见后面的“参考阅读”小节。

5

5.3.2　具体步骤

我们将首先初始化数组，接着生成和绘制分形图，最后把分形图并入图像Lena。

1. 初始化数组。

用`meshgrid`、`zeros`和`linspace`函数，把图像上各个像素对应的`x`、`y`和`z`数组进行初始化。

```
x, y = numpy.meshgrid(numpy.linspace
    (x_min, x_max, SIZE),
    numpy.linspace(y_min, y_max, SIZE))
c = x + 1j * y
z = c.copy()
fractal = numpy.zeros
    (z.shape, dtype=numpy.uint8) +
    MAX_COLOR
```

2. 生成分形图。

如果z是一个复数，则曼德勃罗分形的迭代公式是：

$$z_{n+1} = z_n^2 + c$$

上式中c是复常数。可以用图形化的方式在复平面中表示曼德勃罗分形，实轴对应z的实部，虚轴对应z的虚部。我们将使用逃逸时间算法绘制分形图。

首先选定一个到原点的距离不超过指定值（2左右）的小区域，该算法将扫描这个区域内的各个点。这个区域中的每个点都被用作一个c值，每个点的颜色由该点逃离这个区域所需要的迭代次数决定。如果这个迭代次数超出某个预先设定的值，则该点的颜色就设定为默认的背景色。更多相关的信息请浏览前面提到的维基百科页面。

```
for n in range(ITERATIONS):
    print n
    mask = numpy.abs(z) <= 4
    z[mask] = z[mask] ** 2 +  c[mask]
    fractal[(fractal == MAX_COLOR) &
      (-mask)] = (MAX_COLOR - 1) *
      n / ITERATIONS
```

3. 绘制分形图。

用Matplotlib绘制分形图。

```
matplotlib.pyplot.subplot(211)
matplotlib.pyplot.imshow(fractal)
```

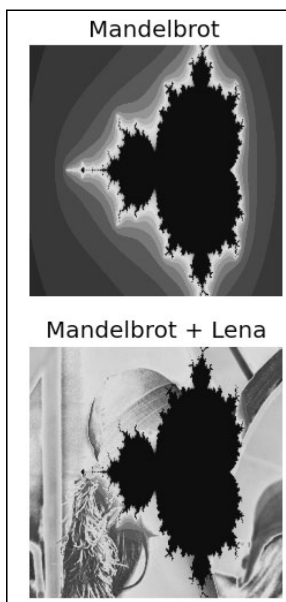
```
matplotlib.pyplot.title('Mandelbrot')
matplotlib.pyplot.axis('off')
```

4. 把分形图和图像Lena合并。

使用`choose`函数，从分形图或图像Lena中选择各点的取值。

```
matplotlib.pyplot.subplot(212)
matplotlib.pyplot.imshow(numpy.choose
  (fractal < lena, [fractal, lena]))
matplotlib.pyplot.axis('off')
matplotlib.pyplot.title
  ('Mandelbrot + Lena')
```

最终生成如下的图形。



本攻略的完整代码如下。

```
import numpy
import matplotlib.pyplot
import sys
import scipy

if(len(sys.argv) != 2):
   print "Please input the number of iterations for the fractal"
   sys.exit()

ITERATIONS = int(sys.argv[1])
lena = scipy.misc.lena()
```

```
SIZE = lena.shape[0]
MAX_COLOR = 255.
x_min, x_max = -2.5, 1
y_min, y_max = -1, 1

# 初始化数组
x, y = numpy.meshgrid(numpy.linspace(x_min, x_max, SIZE),
                      numpy.linspace(y_min, y_max, SIZE))
c = x + 1j * y
z = c.copy()
fractal = numpy.zeros(z.shape, dtype=numpy.uint8) + MAX_COLOR

# 生成分形图
for n in range(ITERATIONS):
    print n
    mask = numpy.abs(z) <= 4
    z[mask] = z[mask] ** 2 +  c[mask]
    fractal[(fractal == MAX_COLOR) & (-mask)] = (MAX_COLOR - 1) * n / ITERATIONS

# 显示分形图
matplotlib.pyplot.subplot(211)
matplotlib.pyplot.imshow(fractal)
matplotlib.pyplot.title('Mandelbrot')
matplotlib.pyplot.axis('off')

# 与图像Lena合并
matplotlib.pyplot.subplot(212)
matplotlib.pyplot.imshow(numpy.choose(fractal < lena, [fractal, lena]))
matplotlib.pyplot.axis('off')
matplotlib.pyplot.title('Mandelbrot + Lena')

matplotlib.pyplot.show()
```

5.3.3 攻略小结

在本攻略中，我们用到了下列函数。

函　　数	功能描述
linspace	返回某个范围内的指定间距的数列
choose	根据给定条件从数组中选择数值并创建一个新的数组
meshgrid	输入一个x坐标数组和一个y坐标数组，返回二维坐标数组

5.3.4 参考阅读

- 1.5节“安装Matplotlib”
- 2.2节“安装SciPy”

5.4 图像的模糊化处理

我们可以用高斯滤波器对图像进行模糊化处理。更多关于高斯滤波器的信息请见http://en.wikipedia.org/wiki/Gaussian_filter。高斯滤波器基于正态分布。在SciPy中有一个对应的高斯滤波器函数，需要用标准差作为输入参数。

在本攻略中，我们还将绘制一条极坐标的玫瑰线（polar rose）和一条螺线（更多有关极坐标系的信息请见http://en.wikipedia.org/wiki/Polar_coordinate_system）。这些图案和图像的模糊化处理并不直接相关，把它们添加到图像中，只是为了更有趣。

5.4.1 具体步骤

为了在极坐标系下绘图，首先要做一些初始化工作。接着将对图像Lena进行模糊化处理，并把它绘制在极坐标平面内。

1. 初始化。

为了在极坐标系下绘图，需要做如下的初始化工作。

```
NFIGURES = int(sys.argv[1])
k = numpy.random.random_integers
  (1, 5, NFIGURES)
a = numpy.random.random_integers
  (1, 5, NFIGURES)

colors = ['b', 'g', 'r', 'c',
  'm', 'y', 'k']
```

2. 对图像Lena进行模糊化处理。

使用高斯滤波器对图像Lena进行模糊化处理，标准差的取值为4。

```
matplotlib.pyplot.subplot(212)
blurred = scipy.ndimage.gaussian_filter
  (lena, sigma=4)

matplotlib.pyplot.imshow(blurred)
matplotlib.pyplot.axis('off')
```

3. 在极坐标系下绘图。

Matplotlib中有一个`polar`函数，用来在极坐标系下绘图。

```
theta = numpy.linspace
      (0, k[0] * numpy.pi, 200)
```

```
    matplotlib.pyplot.polar
      (theta, numpy.sqrt(theta), choice(colors))

    for i in xrange(1, NFIGURES):
      theta = numpy.linspace
        (0, k[i] * numpy.pi, 200)
      matplotlib.pyplot.polar
        (theta, a[i] * numpy.cos(k[i] *
        theta), choice(colors))
```

最终生成如下的图形。



本攻略的完整代码如下。

```
import numpy
import matplotlib.pyplot
from random import choice
import sys
import scipy
import scipy.ndimage

# 初始化
NFIGURES = int(sys.argv[1])
k = numpy.random.random_integers(1, 5, NFIGURES)
a = numpy.random.random_integers(1, 5, NFIGURES)

colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k']
```

```
lena = scipy.misc.lena()
matplotlib.pyplot.subplot(211)
matplotlib.pyplot.imshow(lena)
matplotlib.pyplot.axis('off')

# 对图像Lena进行模糊化处理
matplotlib.pyplot.subplot(212)
blurred = scipy.ndimage.gaussian_filter(lena, sigma=4)

matplotlib.pyplot.imshow(blurred)
matplotlib.pyplot.axis('off')

# 在极坐标系下绘图
theta = numpy.linspace(0, k[0] * numpy.pi, 200)
matplotlib.pyplot.polar(theta, numpy.sqrt(theta), choice(colors))

for i in xrange(1, NFIGURES):
   theta = numpy.linspace(0, k[i] * numpy.pi, 200)
   matplotlib.pyplot.polar(theta, a[i] * numpy.cos(k[i] * theta), choice(colors))

matplotlib.pyplot.axis('off')

matplotlib.pyplot.show()
```

5

5.4.2 攻略小结

在本攻略中，我们用到了下列函数。

函　　数	功能描述
gaussian_filter	应用高斯滤波器
random_integers	返回一个数组，数组元素是给定的上下边界之间的随机整数
polar	在极坐标系下绘图

5.5 复制声音片段

正如我们在第2章所见，只需要下载WAV文件并用SciPy加载，就可以对WAV文件做一些简洁的操作。本攻略中，我们将下载一个WAV文件并把其中包含的声音片段重复三次。我们将略去一些在第2章中已经介绍的步骤。

5.5.1 具体步骤

1. 复制声音片段。

虽然NumPy中有一个`repeat`函数，但在本攻略中选用`tile`函数才是更适当的选择。`repeat`

函数通过把每个元素都单独复制若干次的方式来达到扩展数组的效果，而不是把数组内容作为一个整体进行复制。

如下的IPython会话清楚地展示了这两个函数的区别。

```
In: x = array([1, 2])

In: x
Out: array([1, 2])

In: repeat(x, 3)
Out: array([1, 1, 1, 2, 2, 2])

In: tile(x, 3)
Out: array([1, 2, 1, 2, 1, 2])
```

了解这个知识点后，使用tile函数处理数据。

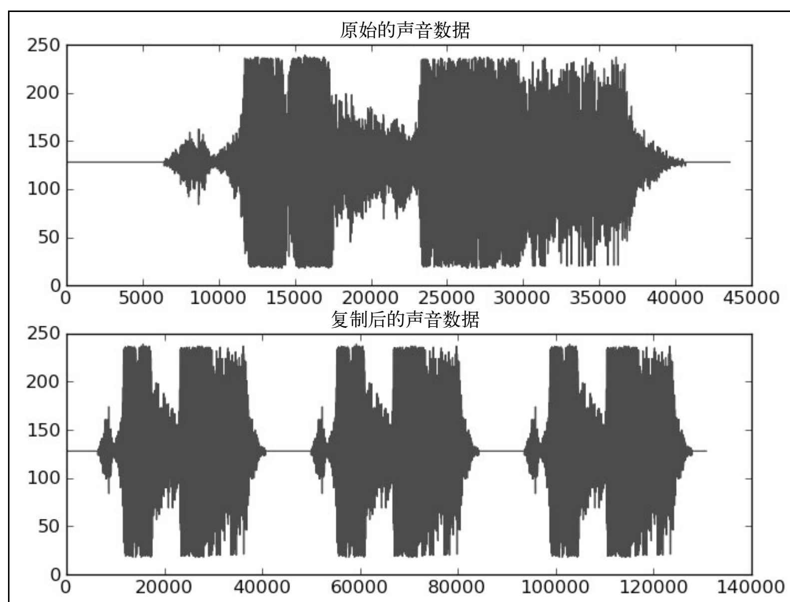
```
repeated = numpy.tile(data, int(sys.argv[1]))
```

2. 绘制声音数据。

用Matplotlib绘制声音数据。

```
matplotlib.pyplot.title("Repeated")
matplotlib.pyplot.plot(repeated)
```

原始的声音数据和复制后的数据如下图所示。



本攻略的完整代码如下。

```
import scipy.io.wavfile
import matplotlib.pyplot
import urllib2
import numpy
import sys

response =
urllib2.urlopen('http://www.thesoundarchive.com/austinpowers/smashingbaby.wav')
print response.info()
WAV_FILE = 'smashingbaby.wav'
filehandle = open(WAV_FILE, 'w')
filehandle.write(response.read())
filehandle.close()
sample_rate, data = scipy.io.wavfile.read(WAV_FILE)
print "Data type", data.dtype, "Shape", data.shape

matplotlib.pyplot.subplot(2, 1, 1)
matplotlib.pyplot.title("Original")
matplotlib.pyplot.plot(data)

matplotlib.pyplot.subplot(2, 1, 2)

# 复制声音片段
repeated = numpy.tile(data, int(sys.argv[1]))

# 绘制声音数据
matplotlib.pyplot.title("Repeated")
matplotlib.pyplot.plot(repeated)
scipy.io.wavfile.write("repeated_yababy.wav",
    sample_rate, repeated)

matplotlib.pyplot.show()
```


5.5.2　攻略小结

本攻略用到的几个最重要的函数如下。

函　　数	功能描述
scipy.io.wavfile.read	把WAV文件加载到数组
numpy.tile	把数组内容重复指定次数
scipy.io.wavfile.write	按照指定的采样率，从NumPy数组创建WAV文件

5.6　合成声音

声音在数学上可以表示为具有特定幅度、频率和相位的正弦波。我们可以从维基百科页面http://en.wikipedia.org/wiki/Piano_key_frequencies提供的列表中随机地选择频率。这些频率值是用如下公式得到的。

$$440 \cdot 2^{\frac{n-49}{12}}$$

上式中的变量n是钢琴中的琴键编号，其取值范围是1~88。幅度、持续时间和相位参数将随机选取。

5.6.1　具体步骤

首先将初始化随机变量，再生成正弦波，然后谱曲，最后用Matplotlib绘制生成的声音数据。

1. 初始化。

随机变量的初始化：

- 幅度值在200到2000之间
- 持续时间在0.01到0.2之间
- 用上述公式选择频率
- 初始相位值在0到2π之间
- NTONES = int(sys.argv[1])

```
amps = 2000. * numpy.random.random
  ((NTONES,)) + 200.
durations = 0.19 * numpy.random.random
  ((NTONES,)) + 0.01
keys = numpy.random.random_integers
  (1, 88, NTONES)
freqs = 440.0 * 2 ** ((keys - 49.)/12.)
phi = 2 * numpy.pi * numpy.random.random((NTONES,))
```

2. 生成正弦波。

编写generate函数，用来生成正弦波。

```
def generate(freq, amp, duration, phi):
  t = numpy.linspace
    (0, duration, duration * RATE)
  data = numpy.sin(2 * numpy.pi *
```

```
    freq * t + phi) * amp

  return data.astype(DTYPE)
```

3. 谱曲。

生成数个声调后，我们需要谱写一段连贯的旋律。我们暂时只是把各个正弦波连接在一起。这不会生成一段好听的旋律，但可以作为一个起点，以后可以做更多的尝试。

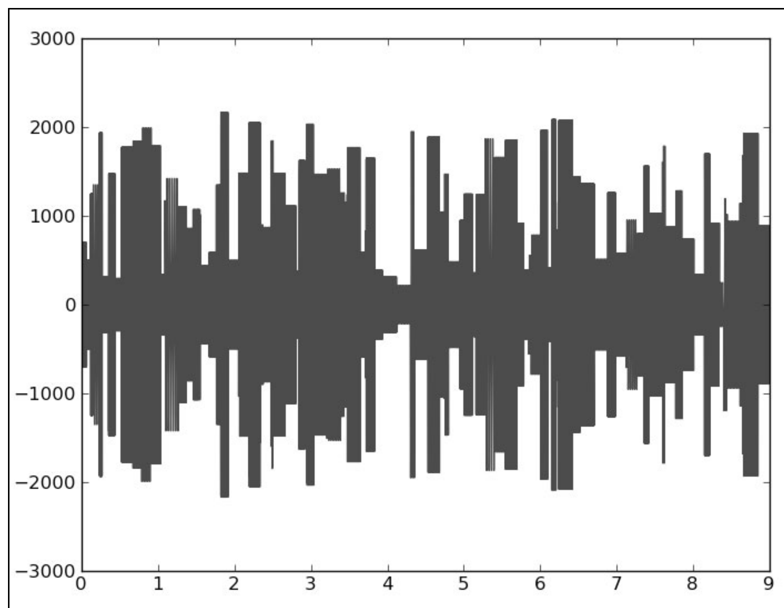
```
for i in xrange(NTONES):
  newtone = generate(freqs[i], amp=amps[i],
    duration=durations[i], phi=phi[i])
  tone = numpy.concatenate((tone, newtone))
```

4. 绘制数据。

用Matplotlib绘制合成的声音数据。

```
matplotlib.pyplot.plot(numpy.linspace
  (0, len(tone)/RATE, len(tone)), tone)
matplotlib.pyplot.show()
```

合成的声音数据如下图所示。



本攻略的完整代码如下。

```
import scipy.io.wavfile
import numpy
```

```
import sys
import matplotlib.pyplot

RATE = 44100
DTYPE = numpy.int16

# 生成正弦波
def generate(freq, amp, duration, phi):
  t = numpy.linspace(0, duration, duration * RATE)
  data = numpy.sin(2 * numpy.pi * freq * t + phi) * amp

  return data.astype(DTYPE)

if len(sys.argv) != 2:
   print "Please input the number of tones to generate"
   sys.exit()

# 初始化
NTONES = int(sys.argv[1])
amps = 2000. * numpy.random.random((NTONES,)) + 200.
durations = 0.19 * numpy.random.random((NTONES,)) + 0.01
keys = numpy.random.random_integers(1, 88, NTONES)
freqs = 440.0 * 2 ** ((keys - 49.)/12.)
phi = 2 * numpy.pi * numpy.random.random((NTONES,))

tone = numpy.array([], dtype=DTYPE)

# 谱曲
for i in xrange(NTONES):
   newtone = generate(freqs[i], amp=amps[i], duration=durations[i], phi=phi[i])
   tone = numpy.concatenate((tone, newtone))

scipy.io.wavfile.write('generated_tone.wav', RATE, tone)

# 绘制声音数据
matplotlib.pyplot.plot(numpy.linspace(0, len(tone)/RATE, len(tone)), tone)
matplotlib.pyplot.show()
```

5.6.2　攻略小结

我们用随机生成的声音创建了一个WAV文件。concatenate函数用来把各个正弦波连接起来。

5.7　设计音频滤波器

我记得当初是在模拟电子技术课上学习了各种类型的滤波器，之后又实际构建了这些滤波器。正如你能想到的，用软件实现一个滤波器比用硬件实现要容易得多。

我们将构建一个滤波器，并把它应用于我们下载的一个音频片段。在本章前面的攻略中，已经介绍了一些相关的基本步骤，此处不再重复介绍。

5.7.1 具体步骤

顾名思义，`iirdesign`函数可以用来构建多种类型的模拟和数字滤波器。scipy.signal模块包含了各种和信号处理有关的函数，其中包括`iirdesign`函数。

1. 设计滤波器。

下面用scipy.signal模块中的`iirdesign`函数设计滤波器。

IIR表示无限冲激响应（Infinite Impulse Response），更多相关的信息请见维基百科页面http://en.wikipedia.org/wiki/Infinite_impulse_response。有关`iirdesign`函数的各种细节，这里不做介绍。如有必要，请查阅相关的文档页面http://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.iirdesign.html。

简而言之，我们将设置如下参数。

- 取值范围在0到1之间的归一化频率
- 通带最大衰减
- 阻带最小衰减
- 滤波器类型

```
b,a = scipy.signal.iirdesign
  (wp=0.2, ws=0.1, gstop=60,
  gpass=1, ftype='butter')
```

上述的配置参数对应的是一个巴特沃思带通滤波器。更多有关巴特沃思带通滤波器的信息请见http://en.wikipedia.org/wiki/Butterworth_filter。

2. 使用滤波器。

使用`scipy.signal.lfilter`函数完成滤波器有关的计算工作。该函数的输入参数是上述步骤中`iirdesign`函数的返回结果和一个待滤波的数组。

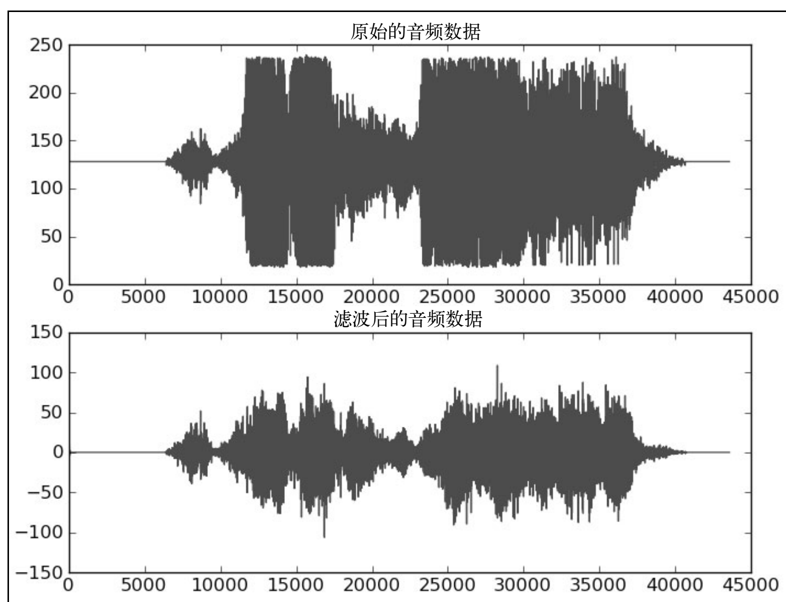
```
filtered = scipy.signal.lfilter(b, a, data)
```

3. 保存新的音频文件。

把滤波后的音频数据写入文件时，需要确保其数据类型与原始的音频数据相同。

```
scipy.io.wavfile.write('filtered.wav',
  sample_rate, filtered.astype(data.dtype))
```

绘制出原始的和滤波后的音频数据后，我们得到如下的图形。



音频滤波器的完整代码如下。

```
import scipy.io.wavfile
import matplotlib.pyplot
import urllib2
import scipy.signal

response = urllib2.urlopen
  ('http://www.thesoundarchive.com/austinpowers/smashingbaby.wav')
print response.info()
WAV_FILE = 'smashingbaby.wav'
filehandle = open(WAV_FILE, 'w')
filehandle.write(response.read())
filehandle.close()
sample_rate, data = scipy.io.wavfile.read(WAV_FILE)
print "Data type", data.dtype, "Shape", data.shape

matplotlib.pyplot.subplot(2, 1, 1)
matplotlib.pyplot.title("Original")
matplotlib.pyplot.plot(data)

# 设计滤波器
b,a = scipy.signal.iirdesign(wp=0.2, ws=0.1, gstop=60, gpass=1, ftype='butter')

# 滤波
```

```
filtered = scipy.signal.lfilter(b, a, data)

# 绘制滤波后的数据
matplotlib.pyplot.subplot(2, 1, 2)
matplotlib.pyplot.title("Filtered")
matplotlib.pyplot.plot(filtered)

scipy.io.wavfile.write('filtered.wav', sample_rate, filtered.astype(data.dtype))

matplotlib.pyplot.show()
```

5.7.2 攻略小结

我们创建和使用了一个巴特沃思带通滤波器。为了创建滤波器，我们使用了下列函数。

函　　数	功能描述
`scipy.signal.iirdesign`	创建一个IIR类型的数字或模拟滤波器。该函数支持丰富的参数选项，相关的文档请见http://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.iirdesign.html
`scipy.signal.lfilter`	用给定的数字滤波器对数组滤波

5.8 用索贝尔滤波器进行边缘检测

索贝尔算子（Sobel operator，更多相关信息请见http://en.wikipedia.org/wiki/Sobel_operator）可被用来做图像的边缘检测。此类边缘检测就是对图像的亮度进行离散差分运算。因为图像是二维的，所以差分运算所得的梯度值也有两个分量。当然我们可以选择只计算某一个维度的梯度值。我们将对图像Lena应用索贝尔滤波器。

5.8.1 具体步骤

我们将以图像Lena为研究对象，学习怎样用索贝尔滤波器进行边缘检测。

1. 在x轴方向应用索贝尔滤波器。

为了在x轴方向应用索贝尔滤波器，需要把参数`axis`设置为0。

```
sobelx = scipy.ndimage.sobel
  (lena, axis=0, mode='constant')
```

2. 在y轴方向应用索贝尔滤波器。

为了在y轴方向应用索贝尔滤波器，需要把参数`axis`设置为1。

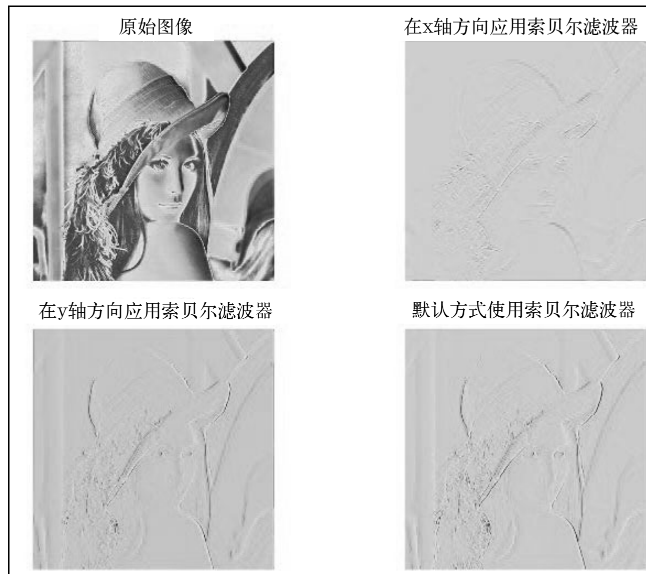
```
sobely = scipy.ndimage.sobel
  (lena, axis=1, mode='constant')
```

3. 默认方式使用索贝尔滤波器。

默认方式使用索贝尔滤波器时，只需要提供一个数组：

```
default = scipy.ndimage.sobel(lena)
```

原始图像和处理后的图像如下图所示。该图展示了用索贝尔滤波器进行边缘检测的效果。



完整的边缘检测代码如下。

```
import scipy
import scipy.ndimage
import matplotlib.pyplot

lena = scipy.misc.lena()

matplotlib.pyplot.subplot(221)
matplotlib.pyplot.imshow(lena)
matplotlib.pyplot.title('Original')
matplotlib.pyplot.axis('off')

# 在x轴方向应用索贝尔滤波器
sobelx = scipy.ndimage.sobel(lena, axis=0, mode='constant')

matplotlib.pyplot.subplot(222)
matplotlib.pyplot.imshow(sobelx)
```

```
matplotlib.pyplot.title('Sobel X')
matplotlib.pyplot.axis('off')

# 在y轴方向应用索贝尔滤波器
sobely = scipy.ndimage.sobel(lena, axis=1, mode='constant')

matplotlib.pyplot.subplot(223)
matplotlib.pyplot.imshow(sobely)
matplotlib.pyplot.title('Sobel Y')
matplotlib.pyplot.axis('off')

# 默认方式使用索贝尔滤波器
default = scipy.ndimage.sobel(lena)

matplotlib.pyplot.subplot(224)
matplotlib.pyplot.imshow(default)
matplotlib.pyplot.title('Default Filter')
matplotlib.pyplot.axis('off')

matplotlib.pyplot.show()
```


5.8.2　攻略小结

我们把索贝尔滤波器应用于图像Lena。正如我们所见，我们可以指定沿某个轴向进行滤波运算。默认设置是不指定轴向的。

第 6 章

特殊类型数组与通用函数

本章主要内容：

- 创建一个通用函数
- 寻找勾股数
- 用chararray做字符串操作
- 创建一个masked类型的数组
- 忽略负值和极值
- 用recarray数组创建评分表

6.1 引言

本章将介绍NumPy的特殊类型数组和通用函数。你未必会天天用到这些内容，但其重要性不容低估。通用函数（Universal functions，简称Ufuncs）可以对数组的每一个元素逐一进行操作，也可以用来对标量进行操作。Ufuncs的输入参数是一组标量，其返回值是另一组标量。加减乘除等各种数学运算都有对应的通用函数。本章介绍的几种特殊类型的数组，都是基本的NumPy数组对象的子类，都提供了额外的功能。

6.2 创建一个通用函数

使用NumPy中的frompyfunc函数，可以利用一个Python函数创建通用函数。

6.2.1 具体步骤

下面是创建一个通用函数的步骤。

1. 定义Python函数。

定义一个简单的Python函数，其功能是把输入值加倍。

```
def double(a):
    return 2 * a
```

2. 创建通用函数。

用`frompyfunc`创建通用函数时，需要指明输入参数的个数和返回对象的个数。

```
import numpy

def double(a):
    return 2 * a

ufunc = numpy.frompyfunc(double, 1, 1)
print "Result", ufunc(numpy.arange(4))
```

上述代码执行后，输出如下结果。

```
Result [0 2 4 6]
```

6.2.2 攻略小结

我们定义了一个Python函数，其功能是把输入值加倍。实际上把字符串作为这个函数的输入值也是可以的，Python语法允许这样做。我们用NumPy中的`frompyfunc`函数，基于这个Python函数创建了一个通用函数。

6.3 寻找勾股数

勾股数即毕氏三元数（Pythagorean triple），有关勾股数的更多信息请见维基百科页面http://en.wikipedia.org/wiki/Pythagorean_triple。勾股数和勾股定理（毕式定理）是紧密相关的。想必你在中学时代的几何课中，已经学过勾股定理了。

构成勾股数的三个自然数分别代表直角三角形的三条边，因此遵守勾股定理。本例将寻找三个自然数的和等于1000的勾股数，这需要用到欧几里得公式：

$$a = m^2 - n^2, b = 2mn, c = m^2 + n^2$$

我们将在本例中看到几个通用函数的实际运用。

6.3.1 具体步骤

欧几里得公式定义了索引`m`和`n`。

1. 创建数组m和数组n。

创建数组，用来存放索引。

```
m = numpy.arange(33)
n = numpy.arange(33)
```

2. 计算勾股数的构成元素a、b和c。

使用欧几里得公式，计算勾股数的构成元素a、b和c。对数组进行求积、求差和求和运算时，需要用到outer函数。

```
a = numpy.subtract.outer(m ** 2, n ** 2)
b = 2 * numpy.multiply.outer(m, n)
c = numpy.add.outer(m ** 2, n ** 2)
```

3. 找到符合要求的索引。

现在已经生成了分别包含勾股数的构成元素a、b和c的三个数组。需要进一步寻找符合要求的勾股数。使用NumPy的where函数，可以获得符合要求的勾股数的索引。

```
idx = numpy.where((a + b + c) == 1000)
```

4. 检查得到的结果。

用numpy.testing模块检查得到的结果。

```
numpy.testing.assert_equal
  (a[idx]**2 + b[idx]**2, c[idx]**2)
```

本攻略的完整代码如下。

```
import numpy
import numpy.testing

#勾股数（毕氏三元数）由三个自然数a、b、c构成，并且满足条件a<b<c
#a ** 2 + b ** 2 = c ** 2
#
#例如，3**2 + 4**2 = 9 + 16 = 25 = 5**2.
#
#有且仅有一个勾股数满足条件a + b + c = 1000。
#找到这个勾股数，并计算其构成元素的乘积。

#1. 创建数组m和数组n
m = numpy.arange(33)
n = numpy.arange(33)

#2. 计算勾股数的构成元素a、b和c
a = numpy.subtract.outer(m ** 2, n ** 2)
```

```
b = 2 * numpy.multiply.outer(m, n)
c = numpy.add.outer(m ** 2, n ** 2)

#3. 找到符合要求的索引
idx =  numpy.where((a + b + c) == 1000)

#4. 检查得到的结果
numpy.testing.assert_equal(a[idx]**2 + b[idx]**2, c[idx]**2)
print a[idx] * b[idx] * c[idx]
```

6.3.2　攻略小结

通用函数Ufuncs不是一般意义上的函数，实际上是具备函数功能的对象。正如我们在本例中所见，Ufuncs对象中有一个outer方法。NumPy中标准的Ufuncs很多是用C语言实现的，因此其运算速度比常规的Python代码要快。Ufuncs支持对元素的逐个处理和类型转换，这实际上意味着能减少循环语句的使用。

6.4　用 chararray 做字符串操作

NumPy中有一个chararray对象，可以专门用来存放字符串。chararray是ndarray的子类，具有专门对字符串进行操作的方法。我们将从Python的官网下载一些文本，并用chararray中的方法对文本进行处理。相对于常规的NumPy数组，使用chararray的优点如下。

- 用索引获取数组元素时，字符串中多余的空格会自动删除。
- 做比较运算时，字符串后端的空格会自动删除。
- 支持向量化的字符串操作，不需要使用循环语句。

6

6.4.1　具体步骤

让我们开始创建chararray数组。

1. 创建chararray数组。

以视图的方式，创建chararray数组。

```
carray = numpy.array(html).view(numpy.chararray)
```

2. 把TAB字符替换为空格。

使用expandtabs函数，把TAB字符替换为空格。使用该函数时，可以通过输入参数指定TAB对应的空格数，默认值是8。

```
carray = carray.expandtabs(1)
```

3. 按行分割字符串。

使用splitlines函数，按行分割字符串。

```
carray = carray.splitlines()
```

本攻略的完整代码如下。

```
import urllib2
import numpy
import re

response = urllib2.urlopen('http://python.org/')
html = response.read()
html = re.sub(r'<.*?>', '', html)
carray = numpy.array(html).view(numpy.chararray)
carray = carray.expandtabs(1)
carray = carray.splitlines()
print carray
```

6.4.2　攻略小结

在本攻略中，我们实际体验了特殊类型数组chararray类的用法。该类提供了若干向量化的字符串操作方法，也提供了便利的针对空格的默认处理。

6.5　创建一个 masked 类型的数组

Masked类型的数组（Masked数组）可用来屏蔽数据集中的缺失值和无效值。numpy.ma模块中的MaskedArray类包含一个屏蔽码（mask）数组，而且是ndarray的子类。在本攻略中，我们把图像Lena作为数据源，并且假设图像中的某些数据已被损毁。最后将绘制原始图像、原始图像的对数值、Masked类型的数组及其对数值。

6.5.1　具体步骤

让我们开始创建Masked类型的数组。

1. 创建屏蔽码数组。

为了创建Masked类型的数组，需要先指定一个屏蔽码数组。随机生成一个屏蔽码数组，数组元素的取值是0或1的一个随机数。[①]

① 屏蔽码数组是一个布尔数组，因此只需要生成0和1两个随机数，分别代表False和True。——译者注

```
random_mask = numpy.random.randint
  (0, 2, size=lena.shape)
```

2. 创建Masked类型的数组。

使用上一步得到的屏蔽码数组，创建Masked类型的数组。

```
masked_array = numpy.ma.array
  (lena, mask=random_mask)
```

本攻略的完整代码如下。

```
import numpy
import scipy
import matplotlib.pyplot

lena = scipy.misc.lena()
random_mask = numpy.random.randint(0, 2, size=lena.shape)

matplotlib.pyplot.subplot(221)
matplotlib.pyplot.title("Original")
matplotlib.pyplot.imshow(lena)
matplotlib.pyplot.axis('off')

masked_array = numpy.ma.array(lena, mask=random_mask)
print masked_array

matplotlib.pyplot.subplot(222)
matplotlib.pyplot.title("Masked")
matplotlib.pyplot.imshow(masked_array)
matplotlib.pyplot.axis('off')

matplotlib.pyplot.subplot(223)
matplotlib.pyplot.title("Log")
matplotlib.pyplot.imshow(numpy.log(lena))
matplotlib.pyplot.axis('off')

matplotlib.pyplot.subplot(224)
matplotlib.pyplot.title("Log Masked")
matplotlib.pyplot.imshow(numpy.log(masked_array))
matplotlib.pyplot.axis('off')

matplotlib.pyplot.show()
```

最终生成的图像如下。



6.5.2 攻略小结

我们把随机生成的屏蔽码数组应用到了一个NumPy数组上。这样做的效果是，如果某个数组元素在屏蔽码数组中对应元素的取值为True，则该数组元素在后续处理中将被忽略。在numpy.ma模块中，有一系列完整的针对Masked数组的操作。本攻略只展示了怎样创建一个Masked类型的数组。

6.6 忽略负值和极值

Masked数组的一个用途是忽略取值为负数的数组元素。例如，求取数组的对数时需要忽略其中的负值。Masked数组的另一个用途是排除极值，此时需要给出极值的上下边界。

在本攻略中，我们将把Masked数组的上述功能应用于对股价数据的处理。下载股价数据的步骤在前面的章节中已经介绍过了，此处不再赘述。

6.6.1 具体步骤

下面将对一个包含负数的数组取对数。

1. 对负数取对数。

首先创建一个包含3的倍数的数组。

```
triples = numpy.arange(0, len(close), 3)
print "Triples", triples[:10], "..."
```

然后创建一个全1数组，其大小和股价数据数组相同。

```
signs = numpy.ones(len(close))
print "Signs", signs[:10], "..."
```

利用第2章中介绍的索引技巧，把`signs`数组中索引值为3的倍数的数组元素置为负数。

```
signs[triples] = -1
print "Signs", signs[:10], "..."
```

最后对包含负值的股价数据数组取对数。

```
ma_log = numpy.ma.log(close * signs)
print "Masked logs", ma_log[:10], "..."
```

对于AAPL股票，将有如下的打印输出。

```
Triples [ 0 3 6 9 12 15 18 21 24 27] ...
Signs [ 1. 1. 1.  1. 1. 1.  1. 1. 1.  1.] ...
Signs [-1. 1. 1. -1. 1. 1. -1. 1. 1. -1.] ...
Masked logs [-- 5.93655586575 5.95094223368 -- 5.97468290742
5.97510711452 --
 6.01674381162 5.97889061623 --] ...
```

2. 忽略极值。

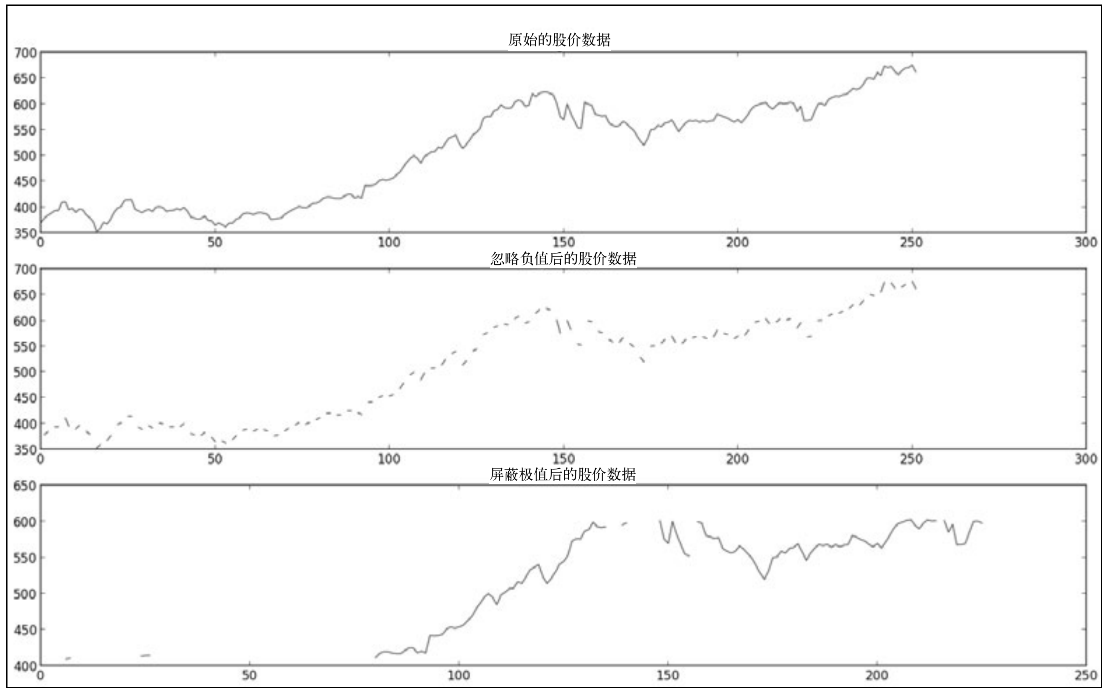
我们把比均值小或大一个标准差以上的数定义为极值。根据这个定义，可以使用如下代码屏蔽极值。

```
dev = close.std()
avg = close.mean()
inside = numpy.ma.masked_outside
  (close, avg - dev, avg + dev)
print "Inside", inside[:10], "..."
```

这段代码会打印出数组`inside`的前10个元素。

```
Inside [-- -- -- -- -- -- 409.429675172
  410.240597855 -- --] ...
```

让我们绘制三组数据：原始的股价数据；对股价数据取对数，再对其结果进行指数运算后，得到的数据；基于标准差屏蔽原始数据中的极值后，得到的数据。最终结果如下图所示。



本攻略的完整代码如下。

```
import numpy
from matplotlib.finance import quotes_historical_yahoo
from datetime import date
import sys
import matplotlib.pyplot

def get_close(ticker):
    today = date.today()
    start = (today.year - 1, today.month, today.day)

    quotes = quotes_historical_yahoo(ticker, start, today)

    return numpy.array([q[4] for q in quotes])

close = get_close(sys.argv[1])

triples = numpy.arange(0, len(close), 3)
print "Triples", triples[:10], "..."

signs = numpy.ones(len(close))
print "Signs", signs[:10], "..."

signs[triples] = -1
```

```
print "Signs", signs[:10], "..."

ma_log = numpy.ma.log(close * signs)
print "Masked logs", ma_log[:10], "..."

dev = close.std()
avg = close.mean()
inside = numpy.ma.masked_outside(close, avg - dev, avg + dev)
print "Inside", inside[:10], "..."

matplotlib.pyplot.subplot(311)
matplotlib.pyplot.title("Original")
matplotlib.pyplot.plot(close)

matplotlib.pyplot.subplot(312)
matplotlib.pyplot.title("Log Masked")
matplotlib.pyplot.plot(numpy.exp(ma_log))

matplotlib.pyplot.subplot(313)
matplotlib.pyplot.title("Not Extreme")
matplotlib.pyplot.plot(inside)

matplotlib.pyplot.show()
```

6.6.2 攻略小结

numpy.ma模块中的函数用来屏蔽不合法的数组元素。例如，`log`函数和`sqrt`函数不允许使用负值。对被屏蔽的值的后续处理，有点像数据库和其他编程语言中的`NULL`。所有对被屏蔽的值的操作，其结果还是一个被屏蔽的值。

6.7 用 recarray 创建评分表

`recarray`类是`ndarray`的子类。可以像在数据库里面一样，用`recarray`存放由不同类型的数据构成的记录。例如，可以用它来存放有关雇员的记录，包括薪资等数值类型的数据以及雇员姓名等字符串类型的数据。

现代经济理论告诉我们，投资可以归结为对风险和回报的优化。风险可以表示为对数收益率的标准差（更多有关算数收益率和对数收益率的信息请见http://en.wikipedia.org/wiki/Rate_of_return#Arithmetic_and_logarithmic_return），而回报可以表示为对数收益率的平均值。我们可以提供一个评分，用相对高的分值表示低风险和高回报。我们将计算若干股票的分值，并以表格的形式，把这些分值和股票代码一起保存到一个NumPy `recarray`对象中。

6.7.1　具体步骤

我们将从创建记录类型的数组开始。

1. 创建记录类型的数组。

创建一个记录类型的数组。每条记录包括股票代码、标准差分值、均值分值和总评分等字段。

```
weights = numpy.recarray((len(tickers),),
  dtype=[('symbol', numpy.str_, 16),
    ('stdscore', float), ('mean', float),
      ('score', float)])
```

2. 初始化分值。

简单起见，我们基于对数收益率，通过一个循环，对分值进行初始化。

```
for i in xrange(len(tickers)):
    close = get_close(tickers[i])
    logrets = numpy.diff
      (numpy.log(close))
    weights[i]['symbol'] =
      tickers[i]
    weights[i]['mean'] =
      logrets.mean()
    weights[i]['stdscore'] =
      1/logrets.std()
    weights[i]['score'] = 0
```

如你所见，可以使用上一步中定义的字段名访问记录中的各个字段。

3. 对分值进行归一化处理。

我们现在已经获得了一些分值，但不同类型的分值是不能直接相互比较的。首先需要对这些分值进行归一化处理，然后才能对其进行组合。归一化意味着要确保同类型的分值加起来等于1。

```
for key in ['mean', 'stdscore']:
    wsum = weights[key].sum()
    weights[key] = weights[key]/wsum
```

4. 计算总评分并排序。

本例中的总评分就是各个归一化处理后的单项分值的平均值。依据总评分对记录进行排序并生成排名表。

```
weights['score'] = (weights
  ['stdscore'] + weights['mean'])/2
weights['score'].sort()
```

本攻略的完整代码如下。

```
import numpy
from matplotlib.finance
import quotes_historical_yahoo
from datetime import date

# 道琼斯工业指数成分股，股利收益率>4%
tickers = ['MRK', 'T', 'VZ']

def get_close(ticker):
    today = date.today()
    start = (today.year - 1, today.month, today.day)

    quotes = quotes_historical_yahoo
      (ticker, start, today)

    return numpy.array([q[4] for q in quotes])

weights = numpy.recarray((len(tickers),),
  dtype=[('symbol', numpy.str_, 16),
  ('stdscore', float), ('mean', float),
  ('score', float)])

for i in xrange(len(tickers)):
    close = get_close(tickers[i])
    logrets = numpy.diff(numpy.log(close))
    weights[i]['symbol'] = tickers[i]
    weights[i]['mean'] = logrets.mean()
    weights[i]['stdscore'] = 1/logrets.std()
    weights[i]['score'] = 0

for key in ['mean', 'stdscore']:
    wsum = weights[key].sum()
    weights[key] = weights[key]/wsum

weights['score'] = (weights['stdscore'] +
  weights['mean'])/2
weights['score'].sort()

for record in weights:
    print "%s,mean=%.4f,stdscore=%.4f,
      score=%.4f" % (record['symbol'],
      record['mean'], record['stdscore'],
      record['score'])
```

程序执行后，输出如下内容。

```
MRK,mean=0.1862,stdscore=0.2886,score=0.2374
T,mean=0.3570,stdscore=0.3556,score=0.3563
VZ,mean=0.4569,stdscore=0.3557,score=0.4063
```

6.7.2　攻略小结

我们计算了几支股票的分值，并把计算结果存储到了一个NumPy recarray对象中。recarray数组允许我们把不同类型的数据（例如本攻略中的股票代码和数值类型的分值）放在一条记录中。recarray数组还允许我们访问数组成员的各个字段，例如arr.field。本攻略介绍了recarray数组的创建。在numpy.recarray模块中，可以发现更多有关recarray数组的功能。

第7章 性能分析与调试

本章主要内容：

- 用timeit进行性能分析
- 用IPython进行性能分析
- 安装line_profiler
- 用line_profiler分析代码
- 用cProfile扩展模块分析代码
- 用IPython进行调试
- 用pudb进行调试

7.1 引言

调试是查找并移除软件中的bug的行为。性能分析（Profiling）是为软件程序构建一个特殊的配置，并在此基础上收集内存使用情况或时间复杂度信息。性能分析与调试是开发者日常工作中不可或缺的活动。对于重要的软件产品而言，尤其如此。好在有很多相关的工具可以为你所用。本章将讨论若干在NumPy用户中比较流行的性能分析与调试技术。

7.2 用 timeit 进行性能分析

Python标准库中的timeit是一个用来测量代码段执行时间的模块。我们将使用不同大小的NumPy数组，测量其调用`sort`函数时所要耗费的时间。经典的快速排序和归并排序算法的平均执行时间是O(nlogn)，因此我们将尝试对测量结果进行拟合，看其是否具有类似的时间复杂度。

7.2.1 具体步骤

我们需要若干用来排序的数组。

1. 创建用来排序的数组。

创建若干不同大小的数组，其中的数组元素都是随机整数。

```
times = numpy.array([])

for size in sizes:
    integers = numpy.random.random_integers
      (1, 10 ** 6, size)
```

2. 测量执行时间。

为了测量时长，需要创建一个定时器，并为其提供一个需要执行的函数和相关的引入语句。然后执行100次排序操作，得到总的排序时间。

```
def measure():
    timer = timeit.Timer('dosort()',
      'from __main__ import dosort')

    return timer.timeit(10 ** 2)
```

3. 构建测量时间数组。

通过逐一添加测量值的方式，构建测量时间数组。

```
times = numpy.append(times, measure())
```

4. 比照nlogn模型拟合数据。

比照nlogn这个理论模型，对测量时间数据进行拟合。因为我们选择的数组大小是2的整数次幂，通过改变幂指数来改变大小，所以相关的计算过程并不复杂。

```
fit = numpy.polyfit(sizes * powersOf2, times, 1)
```

本攻略的完整代码如下。

```
import numpy
import timeit
import matplotlib.pyplot

# 本程序用来测量NumPy中sort函数的性能
# 并绘制出数组大小与执行时间的对应关系图。
integers = []
def dosort():
   integers.sort()

def measure():
   timer = timeit.Timer('dosort()',
     'from __main__ import dosort')

   return timer.timeit(10 ** 2)
```

```
powersOf2 = numpy.arange(0, 19)
sizes = 2 ** powersOf2

times = numpy.array([])

for size in sizes:
   integers = numpy.random.random_integers(1, 10 ** 6, size)
   times = numpy.append(times, measure())

fit = numpy.polyfit(sizes * powersOf2, times, 1)
print fit
matplotlib.pyplot.title("Sort array sizes vs execution times")
matplotlib.pyplot.xlabel("Size")
matplotlib.pyplot.ylabel("(s)")
matplotlib.pyplot.semilogx(sizes, times, 'ro')
matplotlib.pyplot.semilogx(sizes,
  numpy.polyval(fit, sizes * powersOf2))
matplotlib.pyplot.show()
```

最终得到的数组大小和执行时间的对应关系图如下。

7.2.2　攻略小结

我们测量了NumPy中`sort`函数的平均执行时间。本攻略中用到了如下函数。

函　　数	功能描述
`random_integers`	给定取值区间和数组大小，创建一个随机整数数组
`append`	向NumPy数组添加新元素
`polyfit`	把输入数据拟合为指定阶数的多项式曲线
`polyval`	根据给定的x的具体数值，计算并返回多项式的取值
`semilogx`	绘制数据，要求在x轴采用对数坐标

7.3　用 IPython 进行性能分析

在IPython中，既可以用timeit分析一小段代码的性能，也可以在运行脚本的时候对整个脚本进行分析。对这两种性能分析方式，我们都将予以介绍。

7.3.1　具体步骤

首先，我们要测量一个代码片段的执行时间。

1. 测量代码片段的执行时间。

以pylab模式启动IPython。

```
ipython -pylab
```

创建一个包含1000个整数的数组，其取值范围在0到1000之间。

```
In [1]: a = arange(1000)
```

测量在该数组中查找“任何事情的答案”——42所需要的时间。没错，42是“任何事情的答案”。如果你对此心存疑问，请浏览http://en.wikipedia.org/wiki/42_%28number%29。

```
In [2]: %timeit searchsorted(a, 42)
100000 loops, best of 3: 7.58 us per loop
```

2. 分析一个脚本的性能。

我们将分析一个小脚本的性能。该脚本对一系列不同大小的、包含随机数的矩阵求逆。NumPy数组的`.I`属性（注意是大写的I）表示数组的逆矩阵。

```
import numpy

def invert(n):
```

```
  a = numpy.matrix(numpy.random.
    rand(n, n))
  return a.I

sizes = 2 ** numpy.arange(0, 12)

for n in sizes:
  invert(n)
```

可以用如下方式，对该脚本的运行时间进行测量。

```
In [1]: %run -t invert_matrix.py

IPython CPU timings (estimated):
    User :       6.08 s.
  System :       0.52 s.
Wall time:      19.26 s.
```

可以使用p选项对脚本进行性能分析。

```
 In [2]: %run -p invert_matrix.py

 852 function calls in 6.597 CPU seconds

    Ordered by: internal time

    ncalls tottime percall cumtime percall filename:lineno(function)
        12   3.228   0.269   3.228     0.2 {numpy.linalg.lapack_lite.dgesv}
        24   2.967   0.124   2.967   0.124 {numpy.core.multiarray._fastCopyAndTran
-spose}
        12   0.156   0.013   0.156   0.013 {method 'rand' of 'mtrand.RandomState'
objects}
        12   0.087   0.007   0.087   0.007 {method 'copy' of 'numpy.ndarray'objects}
        12   0.069   0.006   0.069   0.006 {method 'astype' of 'numpy.ndarray'
objects}
        12   0.025   0.002   6.304   0.525 linalg.py:404(inv)
        12   0.024   0.002   6.328   0.527 defmatrix.py:808(getI)
         1   0.017   0.017   6.596   6.596 invert_matrix.py:1(<module>)
        24   0.014   0.001   0.014   0.001 {numpy.core.multiarray.zeros}
        12   0.009   0.001   6.580   0.548 invert_matrix.py:3(invert)
        12   0.000   0.000   6.264   0.522 linalg.py:244(solve)
        12   0.000   0.000   0.014   0.001 numeric.py:1875(identity)
         1   0.000   0.000   6.597   6.597 {execfile}
        36   0.000   0.000   0.000   0.000 defmatrix.py:279(__array_finalize__)
        12   0.000   0.000   2.967   0.247 linalg.py:139(_fastCopyAndTranspose)
        24   0.000   0.000   0.087   0.004 defmatrix.py:233(__new__)
        12   0.000   0.000   0.000   0.000 linalg.py:99(_commonType)
        24   0.000   0.000   0.000   0.000 {method '__array_prepare__' of 'numpy.
ndarray' objects}
        36   0.000   0.000   0.000   0.000 linalg.py:66(_makearray)
        36   0.000   0.000   0.000   0.000 {numpy.core.multiarray.array}
```

```
       12    0.000    0.000    0.000    0.000 {method 'view' of 'numpy.ndarray' objects}
       12    0.000    0.000    0.000    0.000 linalg.py:127(_to_native_byte_order)
        1    0.000    0.000    6.597    6.597 interactiveshell.py:2270(safe_execfile)
```

7.3.2　攻略小结

我们用性能分析工具对上述NumPy脚本的运行过程进行了分析。下表对分析工具的输出项目进行了总结。

列	描　述
ncalls	调用次数
tottime	总的函数执行时间
percall	单次调用的执行时间，即tottime/ncalls
cumtime	函数的累计执行时间，包括该函数本身的执行时间、在该函数内部调用其他函数花费的时间和递归调用花费的时间

7.4　安装 line_profiler

line_profiler是NumPy的一名开发者编写的。该模块可以对Python代码进行逐行分析。本攻略将介绍line_profiler的基本安装步骤。

7.4.1　准备工作

你也许需要先安装setuptools，这在前面的攻略中已经介绍了。如有必要，可先查阅后面的“参考阅读”小节。如果想安装line_profiler最新的开发版本，你需要使用Mercurial。怎样安装Mercurial超出了本书的讨论范围，相关的安装步骤请见http://mercurial.selenic.com/wiki/Download。

7.4.2　具体步骤

请选择适合你的安装方式。

▶ 使用easy_install或pip安装

使用如下两条命令之一安装line_profiler。

```
easy_install line_profiler
pip install line_profiler
```

▶ 安装开发版本

可以使用Mercurial签出（check out）源代码。

```
$ hg clone https://bitbucket.org/robertkern/line_profiler
```

签出源代码后，用如下方式构建并安装。

```
$ python setup.py install
```

7.4.3 参考阅读

- 1.2节“安装IPython”

7.5 用 line_profiler 分析代码

安装好line_profiler后，就可以开始分析代码了。

7.5.1 具体步骤

显然，我们需要一些可供分析的代码。

1. 编写可供分析的代码。

我们将编写一个脚本，用来计算一系列不同大小的包含随机数的矩阵的平方。同时要求对应的线程能休眠几秒钟。待分析的函数需要用`@profile`进行标记。

```
import numpy
import time

@profile
def multiply(n):
  A = numpy.random.rand(n, n)
  time.sleep(numpy.random.randint(0, 2))
  return numpy.matrix(A) ** 2

for n in 2 ** numpy.arange(0, 10):
  multiply(n)
```

2. 对代码进行分析。

使用line_profiler分析代码时，键入如下命令。

```
$ kernprof.py -l -v mat_mult.py
Wrote profile results to mat_mult.py.lprof
Timer unit: 1e-06 s

File: mat_mult.py
Function: multiply at line 4
```

```
Total time: 3.19654 s

Line # Hits    Time  Per Hit  % Time  Line Contents
==============================================================
     4                                @profile
     5                                def multiply(n):
     6   10   13461   1346.1     0.4   A = numpy.random.rand(n, n)
     7   10 3000689 300068.9     93.9   time.sleep(numpy.random.randint(0, 2))
     8   10  182386  18238.6     5.7   return numpy.matrix(A) ** 2
```

7.5.2　攻略小结

修饰符`@profile`的作用是告诉line_profile，哪个函数需要分析。下表对line_profile的输出项目进行了解释。

列	描　述
`Line #`	脚本文件中的行号
`Hits`	单行代码的执行次数
`Time`	执行单行代码花费的总时间
`Per Hit`	单次执行某行代码的平均时间
`% Time`	执行单行代码花费的时间在所有行的总执行时间中占的百分比
`Line Contents`	单行代码的内容

7.6　用 cProfile 扩展模块分析代码

cProfile是从Python 2.5版本开始引入的C语言扩展模块。该模块可用来对代码进行确定性性能分析（deterministic profiling）。确定性性能分析意味着对时长的测量是精确的，没有采用抽样的方式。与此形成对照的是统计性能分析（statistical profiling），其特点是利用随机抽样来估算测量结果。我们将用cProfile分析一个小的NumPy程序，其功能是对一个由随机数构成的数组进行转置。

具体步骤

和之前的攻略一样，我们需要一段待分析的代码。

1. 编写待分析的代码。

编写`transpose`函数。该函数创建一个由随机数构成的数组并对其进行转置。

```
def transpose(n):
  random_values = numpy.random.random((n, n))
  return random_values.T
```

2. 运行分析器。

运行分析器时，需要把待分析的函数作为输入参数。

```
cProfile.run('transpose(%d)' %(int(sys.argv[1])))
```

本攻略的完整代码如下。

```
import numpy
import cProfile
import sys

def transpose(n):
   random_values = numpy.random.random((n, n))
   return random_values.T

cProfile.run('transpose(%d)' %(int(sys.argv[1])))
```

对于一个1000 × 1000的数组，我们得到如下输出结果。

```
4 function calls in 0.029 CPU seconds
  Ordered by: standard name
  ncalls tottime percall cumtime percall filename:lineno(function)
       1   0.001   0.001   0.029   0.029 <string>:1(<module>)
       1   0.000   0.000   0.028   0.028 cprofile_transpose.py:5(transpose)
       1   0.000   0.000   0.000   0.000 {method 'disable' of '_lsprof.Profiler'
 objects}
       1   0.028   0.028   0.028   0.028 {method 'random_sample' of 'mtrand.Random
State' objects}
```

输出结果中的各个列和7.3节中使用p选项后的输出相同。

7.7 用 IPython 进行调试

有些事情没人愿意做，调试便是其中之一。但是，掌握调试技术还是非常有必要的。调试可能要耗费数小时时间，而且根据墨菲定律（Murphy's law），你真正能留给调试的时间很可能会很少。因此，你要熟悉自己所用的调试工具，做相关的事情时要有计划、有条理。当你找到并修复一个bug后，应该立即做一个测试，这样至少能避免在相同的地方再经历一次痛苦的调试过程。单元测试将会在下一章中介绍。我们将调试下面这段有bug的代码，其中的一条语句试图访问一个不存在的数组元素。

```
import numpy

a = numpy.arange(7)
print a[8]
```

IPython调试器的功能和标准的Python pdb调试器类似，但增加了TAB键代码补全和语法突出显示等特性。

7.7.1　具体步骤

下面的步骤展示了一个典型的调试过程。

1. 在IPython中运行有bug的脚本。

开启一个IPython shell。用如下命令在IPython中运行有bug的脚本。

```
In [1]: %run buggy.py
------------------------------------------------------------------
IndexError                               Traceback (most recent call last)
.../site-packages/IPython/utils/py3compat.pyc in execfile(fname,*where)
    173         else:
    174             filename = fname
--> 175         __builtin__.execfile(filename, *where)

.../buggy.py in <module>()
      2
      3 a = numpy.arange(7)
----> 4 print a[8]

IndexError: index out of bounds
```

2. 启动调试器。

既然程序已经崩溃，我们可以启动调试器了。错误所在的行会被自动设置一个断点。

```
In [2]: %debug
> .../buggy.py(4)<module>()
      2
      3 a = numpy.arange(7)
----> 4 print a[8]
```

3. 列出代码。

可以使用list（或`l`）命令列出代码。

```
ipdb> list
      1 import numpy
      2
      3 a = numpy.arange(7)
----> 4 print a[8]
```

4. 在当前行评估代码。

我们可以在当前行（即调试器当前指向的行）评估或执行任意的代码。

```
ipdb> len(a)
7

ipdb> print a
[0 1 2 3 4 5 6]
```

5. 查看调用栈。

调用栈包含了正在运行的程序中处于活动状态的函数的信息。可以使用bt命令查看调用栈。

```
ipdb> bt
 .../py3compat.py(175)execfile()
    171             if isinstance(fname, unicode):
    172                 filename = fname.encode(sys.getfilesystemencoding())
    173             else:
    174                 filename = fname
--> 175             __builtin__.execfile(filename, *where)
> .../buggy.py(4)<module>()
      0 print a[8]
```

在调用栈中返回到上一级。

```
ipdb> u
> .../site-packages/IPython/utils/py3compat.py(175)execfile()
    173               else:
    174                   filename = fname
--> 175               __builtin__.execfile(filename, *where)
```

在调用栈中进入到下一级。

```
ipdb> d
> .../buggy.py(4)<module>()
      2
      3 a = numpy.arange(7)
----> 4 print a[8]
```

7.7.2 攻略小结

本攻略介绍了怎样使用IPython调试一个NumPy程序。我们设置了一个断点，并且能定位到调用栈中的适当位置。本攻略用到了下列调试命令。

命　　令	功能描述
list或l	列出源代码
bt	显示调用栈
u	在调用栈中返回到上一级
d	在调用栈中进入到下一级

7.8　用 pudb 进行调试

pudb是一个基于控制台的Python调试器，安装过程简单，且支持可视化全屏操作。pudb支持方向键和vi命令，也能在需要的时候与IPython集成。

具体步骤

我们先介绍pudb的安装步骤。

1. 安装pudb[①]。

为了安装pudb，需要执行如下命令。

```
sudo easy_install pudb
```

2. 启动调试器。

让我们对上一篇攻略中介绍的那个有bug的脚本进行调试。使用如下命令启动调试器。

```
python -m pudb buggy.py
```

pudb调试器用户界面的屏幕截图如下。

```
PuDB 2012.3 - ?:help  n:next  s:step into  b:breakpoint
o:output  t:run to cursor  !:python shell
  import numpy                                  Variables:
                                                numpy: <module 'num
>*a = numpy.arange(7)                             py' from '/Librar
  print a[8]                                    Stack:
                                                >> <module> buggy.p

                                                Breakpoints:
                                                >>  buggy.py:3
```

用户主界面的上方显示的是最重要的调试命令。我们还能在界面中看到被调试的代码、变量、栈和已定义的断点。键入q可以退出大多数的菜单。键入n可以使调试器移到下一行。我们也可以用方向键或vi编辑器风格的j键和k键，把光标移动到适当位置并执行特定的操作，例如通过键入b设置断点。

① pudb目前不直接支持Windows平台，但可以通过Cygwin使用它。——译者注

第 8 章 质量保证

本章主要内容：

- 安装Pyflakes
- 用Pyflakes进行静态分析
- 用Pylint分析代码
- 用Pychecker进行静态分析
- 用docstrings测试代码
- 编写单元测试
- 用模拟对象测试代码
- 基于BDD方式的测试

8.1 引言

与人们普遍的看法相反，质量保证关注的主要是如何防止出现bug，而不是查找bug。我们将讨论两种方法，用来改善代码质量，从而防止问题出现。在本章的第一部分，我们将对已经存在的代码进行静态分析。在第二部分，我们将介绍单元测试，包括对模拟对象和行为驱动开发（BDD）的介绍。

8.2 安装 Pyflakes

Pyflakes是Python的一个代码分析包，可用来分析代码，发现各种潜在的问题，例如：

- 引入但没有用到的模块
- 没有用到的变量

8.2.1 准备工作

如有必要，请先安装pip或easy_install。

8.2.2　具体步骤

从下列安装方式中选择其一，完成Pyflakes的安装。

▶ 使用pip安装

使用pip命令安装Pyflakes。

```
sudo pip install pyflakes
```

▶ 使用easy_install安装

使用easy_install命令安装Pyflakes。

```
sudo easy_install pyflakes
```

▶ 在Linux中安装

Linux中有同名的Pyflakes安装包。例如，在Red Hat Linux中用如下命令安装。

```
sudo yum install pyflakes
```

在Debian/Ubuntu中的安装命令是：

```
sudo apt-get install pyflakes
```

8.3　用 Pyflakes 进行静态分析

我们将对NumPy代码库中的部分代码进行静态分析。为此我们需要先用Git从代码库中签出（check out）代码，然后用Pyflakes对其中的部分代码进行静态分析。

8.3.1　具体步骤

1. 签出代码。

为了签出NumPy代码，我们需要用到Git。怎样安装Git不在本书的讨论范围之内。用来从代码库中提取代码的Git命令如下。

```
git clone git://github.com/numpy/numpy.git numpy
```

也可以选择从网址https://github.com/numpy/numpy下载zip格式的源代码。

2. 对代码进行分析。

完成上一步骤后，我们应该已经创建了一个numpy目录，里面包括了全部的NumPy代码。进

入该目录，然后运行如下命令。

```
$ pyflakes *.py
pavement.py:71: redefinition of unused 'md5' from line 69
pavement.py:88: redefinition of unused 'GIT_REVISION' from line 86
pavement.py:314: 'virtualenv' imported but unused
pavement.py:315: local variable 'e' is assigned to but never used
pavement.py:380: local variable 'sdir' is assigned to but never used
pavement.py:381: local variable 'bdir' is assigned to but never used
pavement.py:536: local variable 'st' is assigned to but never used
setup.py:21: 're' imported but unused
setup.py:27: redefinition of unused 'builtins' from line 25
setup.py:124: redefinition of unused 'GIT_REVISION' from line 118
setupegg.py:17: 'setup' imported but unused
setupscons.py:61: 'numpy' imported but unused
setupscons.py:64: 'numscons' imported but unused
setupsconsegg.py:6: 'setup' imported but unused
```

上述命令对当前目录下所有的Python脚本进行了分析，分析其代码风格，检查其违反PEP-8编码规范的地方。你也可以选择只对单个文件进行分析。

8.3.2　攻略小结

如你所见，用Pyflakes分析代码风格和寻找违反PEP-8编码规范的地方是相当简单的。Pyflakes的另一优势是分析速度快，但它能报告的错误类型是相当有限的。

8.4　用 Pylint 分析代码

Pylint是另一个开源的静态代码分析器，最初由Logilab开发。Pylint比Pyflakes复杂，且允许用户做更多的定制，但分析速度比Pyflakes慢。更多相关信息请见http://www.logilab.org/card/pylint_manual。

在本攻略中，我们还是需要先从Git版本库下载NumPy代码。简明起见，此处省略对该步骤的介绍。

8

8.4.1　准备工作

你可以选择从源代码安装Pylint，但各种依赖关系处理起来比较麻烦，因此最好选择用easy_install或pip安装，安装命令如下。

```
easy_install pylint
sudo pip install pylint
```

8.4.2　具体步骤

我们还是选择对NumPy代码库的根目录中的文件进行分析。和用Pyflakes分析相比，我们得到的输出信息要多很多。因为Pylint打印出了太多文本信息，所以这里只列出其中的一小部分。

```
$ pylint *.py
No config file found, using default configuration
************* Module pavement
C: 60: Line too long (81/80)
C:139: Line too long (81/80)
...
W: 50: TODO
W:168: XXX: find out which env variable is necessary to avoid the
pb with python
W: 71: Reimport 'md5' (imported line 143)
F: 73: Unable to import 'paver'
F: 74: Unable to import 'paver.easy'
C: 79: Invalid name "setup_py" (should match (([A-Z_][A-Z0-
9_]*)|(__.*__))$)
F: 86: Unable to import 'numpy.version'
E: 86: No name 'version' in module 'numpy'
C:149: Operator not followed by a space
if sys.platform =="darwin":
                ^^
C:202:prepare_nsis_script: Missing docstring
W:228:bdist_superpack: Redefining name 'options' from outer scope
(line 74)
C:231:bdist_superpack.copy_bdist: Missing docstring
W:275:bdist_wininst_nosse: Redefining name 'options' from outer
scope (line 74)
...
```

8.4.3　攻略小结

Pylint默认输出纯文本的分析结果，但如有需要，可以指定其输出HTML格式的信息。分析结果是一条条的消息，消息的格式如下。

```
MESSAGE_TYPE: LINE_NUM:[OBJECT:] MESSAGE
```

消息类型有以下几种。

- [R] 重构类型，建议进行重构。
- [C] 惯例类型，违反了代码风格。
- [W] 警告类型，针对小问题的警告消息。

- [E] 错误类型，错误或潜在的bug。
- [F] 致命错误类型，发生了致命错误，因此不能进行进一步的分析。

8.4.4 参考阅读

- 8.3节“用Pyflakes进行静态分析”

8.5 用 Pychecker 进行静态分析

Pychecker是一个老牌的静态分析工具。虽然它现在的开发状态不太活跃，但还是值得我们了解的。截止到编写本书的时候，最新的版本号是0.8.19，最近的更新时间是2011年。Pychecker试图引入每个模块，并对其进行处理。通过对代码进行分析，可以发现各种问题，例如，传递的参数个数不正确，不正确的字符串格式，使用了不存在的方法，等等。本攻略将用Pychecker分析代码。

具体步骤

▶ 从源代码压缩文件安装

从http://pychecker.sourceforge.net/下载tar.gz格式的源代码压缩文件。解压缩后执行如下命令。

```
python setup.py install
```

▶ 使用pip安装

我们也可以选择用pip安装Pychecker。

```
sudo pip install http://sourceforge.net/projects/pychecker/files/
pychecker/0.8.19/pychecker-0.8.19.tar.gz/download
```

▶ 分析代码

采用和前面几个攻略类似的方式，对代码进行分析。命令如下。

```
pychecker *.py
...
Warnings...

...

setup.py:21: Imported module (re) not used
setup.py:27: Module (builtins) re-imported

...
```

8.6 用 docstrings 测试代码

docstrings是嵌入在Python代码中的字符串，其内容看上去有点像交互式的会话。这些字符串可用来检验某些假设，或者仅仅把它们看作是一些范例代码。我们需要用doctest模块运行这些测试。

让我们写一个简单的程序来计算阶乘，但该程序没有覆盖所有可能的边界条件。也就是说，某些测试项目是通不过的。

8.6.1 具体步骤

1. 编写docstring。

编写一个包含两个测试项目的docstring，其中一个测试项目可以通过，另一个通不过。这个docstring看起来像是Python shell中的会话。

```
"""
Test for the factorial of 3 that should pass.
>>> factorial(3)
6

Test for the factorial of 0 that should fail.
>>> factorial(0)
1
"""
```

2. 编写NumPy代码。

编写如下的NumPy代码。

```
return numpy.arange(1, n+1).cumprod()[-1]
```

该行代码将创建一个由顺序加1的数构成的数组，计算其累积连乘，并返回计算结果的最后一个元素。我们有意让上述代码在某些时候出错。

3. 执行测试。

前面已经提到，我们需要使用doctest模块执行测试。

```
doctest.testmod()
```

求阶乘和用docstring做测试的完整范例代码如下。

```
import numpy
import doctest
```

```
def factorial(n):
   """
   Test for the factorial of 3 that should pass.
   >>> factorial(3)
   6

   Test for the factorial of 0 that should fail.
   >>> factorial(0)
   1
   """
   return numpy.arange(1, n+1).cumprod()[-1]

doctest.testmod()
```

使用-v选项可以获得详细的输出结果，如下所示。

```
python docstringtest.py -v
Trying:
    factorial(3)
Expecting:
    6
ok
Trying:
    factorial(0)
Expecting:
    1
**********************************************************************
File "docstringtest.py", line 11, in __main__.factorial
Failed example:
    factorial(0)
Exception raised:
    Traceback (most recent call last):
      File ".../doctest.py", line 1253, in __run
        compileflags, 1) in test.globs
      File "<doctest __main__.factorial[1]>", line 1, in <module>
        factorial(0)
      File "docstringtest.py", line 14, in factorial
        return numpy.arange(1, n+1).cumprod()[-1]
    IndexError: index out of bounds
1 items had no tests:
    __main__
**********************************************************************
1 items had failures:
   1 of   2 in __main__.factorial
2 tests in 2 items.
1 passed and 1 failed.
***Test Failed*** 1 failures.
```

8.6.2　攻略小结

如你所见，我们没有考虑对0和负数求阶乘的情况。实际上，由于出现了数组为空的情况，我们得到了一个索引超出边界（index out of bounds）错误。当然这个错误很容易修正，我们将在下一攻略中完成此事。

8.7　编写单元测试

测试驱动开发（TDD）是本世纪以来在软件开发领域出现的一项最重要的技术。TDD的一个非常引人注目的地方，是对单元测试的几近狂热的关注。

单元测试用来对一小段代码（通常是一个函数或方法）进行自动化的测试。Python提供了用于单元测试的PyUnit API。对于NumPy使用者，还可以利用numpy.testing模块提供的各种便捷函数（convenience function）。正如它的名字所示，numpy.testing模块就是专门用来做测试的。

8.7.1　具体步骤

让我们先编写一段代码，把它作为测试对象。

1. 编写阶乘函数。

编写如下的阶乘函数。

```
def factorial(n):
  if n == 0:
    return 1

  if n < 0:
    raise ValueError, "Don't be so negative"

  return numpy.arange(1, n+1).cumprod()
```

计算阶乘的代码和上一攻略相同，但增加了对边界条件的检查。

2. 编写单元测试。

下面将编写单元测试。也许你已经发现，本书没有多少和编写类有关的内容。对于大多数攻略而言，看似没有必要这样做。

让我们做个改变，编写一个类。该类将包含单元测试的内容。该类继承自unittest模块中的`TestCase`类，而unittest模块是Python标准库的一个组成部分。我们用下列参数对阶乘函数的调用过程进行测试。

- 正整数，所谓的愉快路径（happy path）
- 边界条件0
- 负整数，将引起错误

```
class FactorialTest(unittest.TestCase):
    def test_factorial(self):
      #对3的阶乘进行测试，应该能通过。
      self.assertEqual(6, factorial(3)[-1])
      numpy.testing.assert_equal(numpy.array([1, 2, 6]), factorial(3))

     def test_zero(self):
       #对0的阶乘进行测试，应该能通过。
       self.assertEqual(1, factorial(0))

     def test_negative(self):
       #对负整数的阶乘进行测试，应该不能通过。
       #阶乘函数会抛出一个ValueError类型的异常，
       #但我们期望得到一个IndexError类型的异常。
       self.assertRaises(IndexError, factorial(-10))
```

求阶乘并对其进行单元测试的完整代码如下。

```
import numpy
import unittest

def factorial(n):
    if n == 0:
      return 1

    if n < 0:
      raise ValueError, "Don't be so negative"

    return numpy.arange(1, n+1).cumprod()

class FactorialTest(unittest.TestCase):
    def test_factorial(self):
      #对3的阶乘进行测试，应该能通过。
      self.assertEqual(6, factorial(3)[-1])
      numpy.testing.assert_equal(numpy.array([1, 2, 6]), factorial(3))

    def test_zero(self):
      #对0的阶乘进行测试，应该能通过。
      self.assertEqual(1, factorial(0))

    def test_negative(self):
      #对负整数的阶乘进行测试，应该不能通过。
      #阶乘函数会抛出一个ValueError类型的异常，但我们期望得到一个IndexError类型的异常。
      self.assertRaises(IndexError, factorial(-10))

if __name__ == '__main__':
    unittest.main()
```

从下面的运行结果可以看出，对负整数求阶乘的测试没有通过。

```
.E.
======================================================================
ERROR: test_negative (__main__.FactorialTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "unit_test.py", line 26, in test_negative
    self.assertRaises(IndexError, factorial(-10))
  File "unit_test.py", line 9, in factorial
    raise ValueError, "Don't be so negative"
ValueError: Don't be so negative

----------------------------------------------------------------------
Ran 3 tests in 0.001s

FAILED (errors=1)
```

8.7.2　攻略小结

我们介绍了怎样使用Python标准库中的unittest模块实现简单的单元测试。我们编写了一个测试类，该类继承自unittest模块中的`TestCase`类。如下的函数被用来做各种测试。

函　　数	功能描述
`numpy.testing.assert_equal`	测试两个NumPy数组是否相等
`unittest.assertEqual`	测试两个值是否相等
`unittest.assertRaises`	测试是否有某个异常被抛出

NumPy的testing包中包括了若干测试函数，我们应该了解其功能。

函　　数	功能描述
`assert_almost_equal`	如果两个数不能在给定精度的条件下近似相等，则引发一个异常
`assert_approx_equal`	如果两个数不能在给定有效数字位数的条件下近似相等，则引发一个异常
`assert_array_almost_equal`	如果两个数组不能在给定精度的条件下近似相等，则引发一个异常
`assert_array_equal`	如果两个数组不相等，则引发一个异常
`assert_array_less`	如果两个数组的形状不同，或者不满足“第一个数组中的每个元素都比第二个数组中的对应元素小”，则引发一个异常
`assert_raises`	用规定的参数调用一个`callable`对象时，如果没有引发指定类型的异常，就认为测试没通过
`assert_warns`	如果指定类型的警告没有被抛出，就认为测试没通过
`assert_string_equal`	断言两个字符串相等

8.8　用模拟对象测试代码

模拟对象（mock）是真实对象的替代物，用来测试真实对象的部分行为。如果你看过电影《外星人入侵》（*Body Snatchers*），应该已经对模拟对象有基本的了解了。一般来说，只有在不方便创建真实对象（例如数据库连接）时，或者对真实对象的测试会产生不希望有的副作用时，才需要用到模拟对象。例如，我们可能不希望在测试过程中，把数据写入到文件系统或数据库中。

在本攻略中，我们将测试一个核反应堆——当然不是真的，它只是一个类。这个核反应堆类可以做阶乘的计算。假设求阶乘会引起一系列的反应，导致发生核灾难的后果。我们将使用mock包，用一个模拟对象来模拟阶乘计算。

8.8.1　具体步骤

首先我们将安装mock包，然后创建一个模拟对象并用它来测试一段代码。

1. 安装mock。

为了安装mock包，请执行如下的命令。

```
sudo easy_install mock
```

2. 创建一个模拟对象。

核反应堆类有一个`do_work`方法，该方法会调用危险的`factorial`方法。我们希望用模拟对象替代`factorial`方法。用如下方式创建模拟对象。

```
reactor.factorial = MagicMock(return_value=6)
```

这将确保模拟对象被调用后，其返回值是6。

3. 断言行为。

我们可以用若干种方法，检查模拟对象的行为，进而测试真实对象的行为。例如，我们可以断言，我们用正确的参数调用了潜在的有爆炸危险的`factorial`方法，具体如下所示。

```
reactor.factorial.assert_called_with(3, "mocked")
```

使用了模拟对象的完整的测试代码如下。

```
from mock import MagicMock
import numpy
import unittest

class NuclearReactor():
```

```
    def __init__(self, n):
      self.n = n

    def do_work(self, msg):
      print "Working"

      return self.factorial(self.n, msg)

    def factorial(self, n, msg):
      print msg

      if n == 0:
         return 1

      if n < 0:
         raise ValueError, "Core meltdown"

      return numpy.arange(1, n+1).cumprod()

class NuclearReactorTest(unittest.TestCase):
    def test_called(self):
      reactor = NuclearReactor(3)
      reactor.factorial = MagicMock(return_value=6)
      result = reactor.do_work("mocked")
      self.assertEqual(6, result)
      reactor.factorial.assert_called_with(3, "mocked")

    def test_unmocked(self):
      reactor = NuclearReactor(3)
      reactor.factorial(3, "unmocked")
      numpy.testing.assert_raises(ValueError)

if __name__ == '__main__':
    unittest.main()
```

我们向`factorial`方法传递了一个字符串，目的是说明模拟对象并不执行真实对象中的代码。单元测试的作用和上一攻略中介绍的相同。第二个测试没有对核反应堆类做任何测试，只是为了演示不使用模拟对象而直接执行真实代码的情况。

测试后的输出结果如下。

```
Working
.unmocked
.
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

8.8.2　攻略小结

模拟对象不具有任何真实的行为。他们就像是伪装成人类的外星人，但还达不到外星人的智慧程度。正如外星人不能说出他所替换的真实人的生日一样，我们需要对模拟对象进行设置，使其能以适当的方式作出反应。例如，本例中的模拟对象返回6。我们可以记录模拟对象的调用情况——被调用了多少次，用的什么参数。

8.9　基于 BDD 方式的测试

行为驱动开发（BDD）是另一个你可能已经碰到过的流行词汇。使用BDD时，我们首先需要依据某些约定和规则，用英语描述被测系统的预期行为。在本攻略中，我们将看到此类约定的一个例子。

BDD方式背后隐藏的想法是，让不会编程的人，能够以某种方式，编写测试代码的主体部分。这些人可以编写功能点（feature），即包括若干步骤的语句。每个步骤差不多都可以与我们编写的单元测试中的内容（例如用NumPy编写的测试语句）对应。目前存在很多Python的BDD框架。本攻略将使用Lettuce框架测试阶乘函数。

8.9.1　具体步骤

下面将介绍怎样安装Lettuce、建立测试环境和编写测试文档。

1. 安装Lettuce。

为了安装Lettuce，请运行下列两条命令之一。

```
pip install lettuce
sudo easy_install lettuce
```

2. 建立测试环境。

用Lettuce测试阶乘函数，需要按要求组织目录结构。在tests目录中，需要有一个features目录。在features目录中，要包含一个factorial.feature文件和一个steps.py文件。steps.py文件的内容是功能描述和测试代码。

```
./tests:
features

./tests/features:
factorial.feature  steps.py
```

3. 编写测试文档。

提炼业务需求是一项艰巨的工作，而把它们用适合测试的方式表达出来就更难了。幸好本攻略的业务需求相当简单，我们只需要把不同的输入值和它们对应的期望的输出值写出来就可以。

我们需要定义不同的场景（scenario），每个场景包括Given、When和Then三个不同类型的区段。每种类型的区段对应不同的测试步骤。我们为阶乘功能点定义了下列三个场景。

```
Feature: Compute factorial

    Scenario: Factorial of 0
      Given I have the number 0
      When I compute its factorial
      Then I see the number 1

    Scenario: Factorial of 1
      Given I have the number 1
      When I compute its factorial
      Then I see the number 1

    Scenario: Factorial of 3
      Given I have the number 3
      When I compute its factorial
      Then I see the number 1, 2, 6
```

4. 定义步骤。

与上述场景中描述的测试步骤对应，我们将定义若干方法。需要特别注意用来标记方法的文本，其内容和场景描述中的区段内容是匹配的。可以用正则表达式从区段内容中提取出需要的输入参数。

在前面两个场景中，我们需要匹配数字。在最后一个场景中，我们需要匹配字符串。NumPy中的`fromstring`函数用来从字符串创建一个NumPy数组，该字符串由逗号分隔的整数构成。下面的代码用来对上述场景进行测试。

```
from lettuce import *
import numpy

@step('I have the number (\d+)')
def have_the_number(step, number):
    world.number = int(number)

@step('I compute its factorial')
def compute_its_factorial(step):
    world.number = factorial(world.number)

@step('I see the number (.*)')
def check_number(step, expected):
    expected = numpy.fromstring(expected, dtype=int, sep=',')
```

```
    numpy.testing.assert_equal(world.number, expected, \
        "Got %s" % world.number)

def factorial(n):
   if n == 0:
      return 1

   if n < 0:
      raise ValueError, "Core meltdown"

   return numpy.arange(1, n+1).cumprod()
```

5. 执行测试。

执行测试时，首先进入tests目录，然后键入如下命令。

```
$ lettuce

Feature: Compute factorial        # \features\factorial.feature:1

  Scenario: Factorial of 0        # \features\factorial.feature:3
    Given I have the number 0     # \features\steps.py:5
    When I compute its factorial  # \features\steps.py:9
    Then I see the number 1       # \features\steps.py:13

  Scenario: Factorial of 1        # \features\factorial.feature:8
    Given I have the number 1     # \features\steps.py:5
    When I compute its factorial  # \features\steps.py:9
    Then I see the number 1       # \features\steps.py:13

  Scenario: Factorial of 3        # \features\factorial.feature:13
    Given I have the number 3     # \features\steps.py:5
    When I compute its factorial  # \features\steps.py:9
    Then I see the number 1, 2, 6 # \features\steps.py:13

1 feature (1 passed)
3 scenarios (3 passed)
9 steps (9 passed)
```


8.9.2 攻略小结

我们定义了一个包含三个场景的功能点，每个场景都包括若干步骤。我们使用NumPy.testing模块中的函数对相关的步骤进行了测试，并使用`fromstring`函数利用特定格式的字符串创建了一个NumPy数组。

第 9 章 用Cython为代码提速

本章主要内容：

- 安装Cython
- 构建Hello World程序
- 在Cython中使用NumPy
- 调用C语言函数
- 分析Cython代码
- 用Cython求阶乘的近似值

9.1 引言

Cython是一种基于Python的比较新的编程语言。它和Python的不同之处在于我们可以选择声明静态类型。很多编程语言，例如C语言，是静态类型的语言，这意味着我们需要指明变量类型、函数参数和返回类型。此外，C是编译型语言，而Python是解释型语言。按照经验规则，我们可以这样说：C语言比较快，而Python比较灵活。我们可以从Cython代码生成C或C++代码，之后还可以把生成的代码进一步编译为Python的扩展模块。

在本章中，我们将学习和Cython有关的内容。我们将编写一些简单的、能和NumPy协同工作的Cython程序。此外还将对Cython代码进行性能分析。

9.2 安装 Cython

使用Cython之前，需要先安装Cython。Enthought和Sage的发行版本中已经包括了Cython，更多的相关信息请见http://www.enthought.com/products/epd.php和http://sagemath.org/。这里将不会讨论怎样安装这些发行版。此外，我们显然还需要一个C编译器，由Cython生成C语言代码后需要用它进行编译。在某些操作系统上（例如Linux），C编译器已经预先安装好了。本攻略假设你已经安装好了C编译器。

具体步骤

从下列安装方式中选择其一，完成Cython的安装。

▶ 从压缩文件安装

执行如下的步骤，从压缩文件安装Cython。

(1) 从http://cython.org/#download下载源代码压缩文件。

(2) 对下载的文件解压缩。

(3) 使用cd命令进入源代码所在的目录。

(4) 执行如下命令。

```
python setup.py install
```

▶ 使用easy_install或pip安装

可以使用`easy_install cython`或`sudo pip install cython`，从PyPI资源库安装Cython。

▶ 用Windows版本的安装文件安装

在Windows平台上安装Cython时，可以访问http://www.lfd.uci.edu/~gohlke/pythonlibs/#cython，使用非官方的Windows版本的安装文件。

9.3　构建 Hello World 程序

依循编程语言传统的介绍方式，我们的Cython之旅将从一个Hello World范例程序开始。不同于Python，我们需要对Cython代码进行编译。我们将首先编写一个.pyx文件，再由该文件生成C语言代码，然后对.c文件进行编译，最后把编译后生成的模块引入Python程序。

9.3.1　具体步骤

下面将具体介绍怎样构建一个Cython版本的Hello World程序。

1. 编写hello.pyx文件。

首先需要编写一段相当简单的代码，用来打印“Hello World”。这是一段标准的Python代码，只是其所在文件的扩展名是pyx。

```
def say_hello():
  print "Hello World!"
```

2. 编写distutils setup.py脚本。

需要创建一个名为setup.py的文件，用来帮助我们构建Cython代码。

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext

ext_modules = [Extension("hello", ["hello.pyx"])]

setup(
      name = 'Hello world app',
      cmdclass = {'build_ext': build_ext},
      ext_modules = ext_modules
    )
```

如你所见，我们需要在这个文件中指明上一步骤编写的Cython程序名称并给应用一个名字。

3. 构建扩展模块。

使用如下命令构建。

```
python setup.py build_ext --inplace
```

这将首先生成C语言代码，再针对所在平台对其进行编译后，产生如下输出结果。

```
running build_ext
cythoning hello.pyx to hello.c
building 'hello' extension
creating build
```

现在可以使用如下语句，在Python程序中引入刚刚生成的模块。

```
from hello import say_hello
```

9.3.2　攻略小结

在本攻略中，我们介绍了一个传统的Hello World范例。Cython是编译型的语言，需要对Cython代码进行编译。我们编写了一个.pyx文件，其中包含了打印Hello World的相关代码。我们也编写了一个setup.py文件，用来生成和构建C语言程序。

9.4　在 Cython 中使用 NumPy

在Cython中使用NumPy代码和使用Python代码的方法是一样的。我们将介绍一个例子，用来分析一支股票的上涨日所占的比例。上涨日的基本特征就是当日股票的收盘价高于前一交易日。我们将计算二项分布总体率的置信区间（binomial proportion confidence）。有关二项分布总体率的

置信区间的更多信息，请参考http://en.wikipedia.org/wiki/Binomial_proportion_confidence_interval。这个置信区间用来表征上涨日出现概率的可信程度。

9.4.1 具体步骤

下面详细介绍怎样在Cython中使用NumPy代码。为此，需要执行以下步骤。

1. 编写.pyx文件。

编写一个.pyx文件，其中包括一个函数，用来计算上涨日的比例值及其置信区间。该函数首先要计算股价的差值，然后要对大于0的差值进行计数并算出上涨日的比例，最后要应用求置信区间的公式（详见上述维基百科页面）。

```
import numpy

def pos_confidence(numbers):
diffs = numpy.diff(numbers)
n = float(len(diffs))
p = len(diffs[diffs > 0])/n
confidence = numpy.sqrt(p * (1 - p)/ n)

return (p, confidence)
```

2. 编写setup.py文件。

我们把上一攻略中的setup.py文件作为模版，在此基础上进行修改。有几个明显需要修改的地方，例如.pyx文件的名字。

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext

ext_modules = [Extension("binomial_proportion", ["binomial_proportion.pyx"])]

setup(
        name = 'Binomial proportion app',
        cmdclass = {'build_ext': build_ext},
        ext_modules = ext_modules
     )
```

现在可以进行构建了，具体步骤请见上一攻略。

3. 使用Cython模块。

完成构建后，把生成的Cython模块引入Python程序。我们将编写一个用Matplotlib下载股票数据的Python程序，该程序在分析收盘价数据时将用到`confidence`函数。

```
from matplotlib.finance import quotes_historical_yahoo
from datetime import date
import numpy
import sys
from binomial_proportion import pos_confidence

#1. 获取收盘价数据。
today = date.today()
start = (today.year - 1, today.month, today.day)

quotes = quotes_historical_yahoo(sys.argv[1], start, today)
close = numpy.array([q[4] for q in quotes])
print pos_confidence(close)
```

以AAPL为命令行参数，程序运行后的输出结果如下。

```
(0.56746031746031744, 0.031209043355655924)
```

9.4.2　攻略小结

我们计算了AAPL股票上涨日出现的概率及其对应的置信区间。我们在.pyx文件中使用了NumPy代码。采用和上一攻略相同的方式，我们用这个.pyx文件构建出一个Cython模块。最后在Python代码中引入并使用了这个Cython模块。

9.5　调用 C 语言函数

我们可以在Cython中调用C语言函数。作为示例，在本攻略中我们将调用C语言的`log`函数。该函数只能对单个数字进行操作，而NumPy中的`log`函数还可以对数组进行操作。我们将计算股价的对数收益率。

9.5.1　具体步骤

从编写Cython代码开始。

1. 编写.pyx文件。

首先需要从libc命名空间引入C语言的`log`函数。然后通过使用`for`循环，把该函数应用于数组的各个元素。最后使用NumPy中的`diff`函数，计算取对数后所得结果的一阶差分。

```
from libc.math cimport log
import numpy

def logrets(numbers):
```

```
    logs = [log(x) for x in numbers]
    return numpy.diff(logs)
```

在前面的攻略中，我们已经介绍了Cython模块的构建过程。我们只需要对setup.py文件稍作修改即可。

2. 绘制对数收益率。

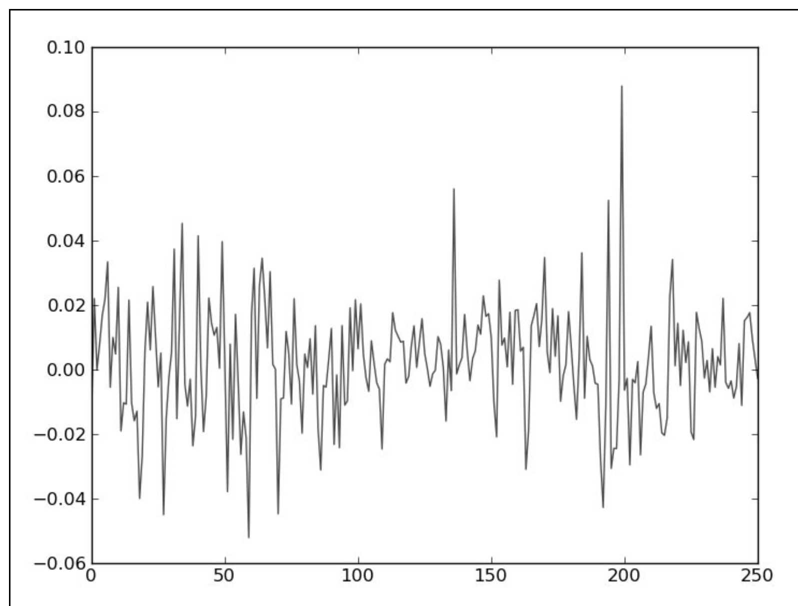
用Matplotlib下载股价数据，把刚刚生成的Cython模块中的`logrets`函数应用于收盘价数据并绘制出结果。

```
from matplotlib.finance import quotes_historical_yahoo
from datetime import date
import numpy
import sys
from log_returns import logrets
import matplotlib.pyplot

today = date.today()
start = (today.year - 1, today.month, today.day)

quotes = quotes_historical_yahoo(sys.argv[1], start, today)
close =  numpy.array([q[4] for q in quotes])
matplotlib.pyplot.plot(logrets(close))
matplotlib.pyplot.show()
```

AAPL股票的对数收益率图示如下。



9.5.2　攻略小结

我们在Cython代码中，调用了C语言的`log`函数。该函数和NumPy中的函数一起用来计算股票的对数收益率。通过这种方式，我们可以创建自己专用的包含便捷函数的API。在Cython中调用C语言函数的好处是，代码的执行速度应该和C语言代码差不多，而代码总体来看还是Python代码。

9.6　分析 Cython 代码

我们将分析用来计算欧拉常数近似值的Cython代码和NumPy代码。所需的计算公式请见http://en.wikipedia.org/wiki/E_%28mathematical_constant%29。

9.6.1　具体步骤

下面介绍怎样分析Cython代码。为此，请执行以下步骤。

▶ 用NumPy近似计算e

用NumPy近似计算e时，需要执行下列步骤。

1. 创建一个由从1到*n*–1的自然数构成的数组（本例中*n*的默认值是40）。
2. 计算数组的累积连乘，等同于求阶乘。
3. 求取阶乘的倒数。
4. 应用上述维基百科页面中的公式。

最后要加上标准的性能分析代码。具体代码如下。[①]

```
import numpy
import cProfile
import pstats

def approx_e(n=40):
   # 创建数组[1, 2, ... n-1]
   arr = numpy.arange(1, n)

   # 计算阶乘，并把计算结果转换为浮点数。
   arr = arr.cumprod().astype(float)

   # 取倒数1/n
   arr = numpy.reciprocal(arr)

   print 1 + arr.sum()
```

① 这段代码在Windows平台上运行报错，原因是，默认情况下数组arr的数据类型是int32，不能容纳太大的计算结果。因此建议创建数组时，使用语句`arr = numpy.arange(1, n, dtype=numpy.uint64)`。——译者注

```
cProfile.runctx("approx_e()", globals(), locals(), "Profile.prof")

s = pstats.Stats("Profile.prof")
s.strip_dirs().sort_stats("time").print_stats()
```

常数e近似计算的结果和性能分析的结果如下所示。更多有关分析结果的信息，请参阅第7章的内容。

```
2.71828182846

         7 function calls in 0.000 CPU seconds

Ordered by: internal time

ncalls tottime percall cumtime percall filename:lineno(function)
     1   0.000   0.000   0.000   0.000 numpy_approxe.py:5(approx_e)
     1   0.000   0.000   0.000   0.000 {method 'cumprod' of 'numpy.ndarray' objects}
     1   0.000   0.000   0.000   0.000 {numpy.core.multiarray.arange}
     1   0.000   0.000   0.000   0.000 {method 'sum' of 'numpy.ndarray' objects}
     1   0.000   0.000   0.000   0.000 {method 'astype' of 'numpy.ndarray' objects}
     1   0.000   0.000   0.000   0.000 <string>:1(<module>)
     1   0.000   0.000   0.000   0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

▶ 用Cython近似计算e

和上述用NumPy近似计算e的过程一样，Cython代码使用相同的算法，只是在具体实现上有一些差别。和NumPy相比，由于缺少合适的便捷函数，我们需要用到`for`循环。而且，我们需要指明某些变量的类型。.pyx文件的具体代码如下。

```
def approx_e(int n=40):
    cdef double sum = 0.
    cdef double factorial = 1.
    cdef int k
    for k in xrange(1,n+1):
        factorial *= k
        sum += 1/factorial
    print 1 + sum
```

下列Python程序首先引入了Cython模块，然后做了一些相关的性能分析。

```
import pstats
import cProfile
import pyximport
pyximport.install()

import approxe
cProfile.runctx("approxe.approx_e()", globals(), locals(), "Profile.prof")
```

```
s = pstats.Stats("Profile.prof")
s.strip_dirs().sort_stats("time").print_stats()
```

Cython代码的分析结果如下。

```
2.71828182846

         3 function calls in 0.000 CPU seconds

   Ordered by: internal time

   ncalls tottime percall cumtime percall filename:lineno(function)
        1   0.000   0.000   0.000   0.000 {approxe.approx_e}
        1   0.000   0.000   0.000   0.000 <string>:1(<module>)
        1   0.000   0.000   0.000   0.000 {method 'disable'of '_lsprof.Profiler' objects}
```

9.6.2　攻略小结

我们分别分析了NumPy和Cython代码。NumPy在执行速度方面已经做了深度优化，NumPy程序和Cython程序都表现出了良好的性能，对此我们不应觉得意外。

9.7　用 Cython 求阶乘的近似值

本章的最后一个攻略是用Cython求阶乘的近似值。我们将使用两种求阶乘近似值的方法。首先使用的是斯特灵近似方法（详见http://en.wikipedia.org/wiki/Stirling%27s_approximation），其计算公式是：

$$\sqrt{2\pi n}(\frac{n}{e})^n$$

然后将使用Ramanujan提出的近似计算方法，其计算公式为：

$$\sqrt{\pi}(\frac{n}{e})^n\sqrt[6]{8n^3+4n^2+n+\frac{1}{30}}$$

9.7.1　具体步骤

下面将具体介绍怎样用Cython计算阶乘的近似值。我们将用到类型声明。你也许还记得，在Cython中，类型声明是可选的。从理论上讲，声明静态类型会提高代码的执行速度。声明静态类型也带来了编写常规的Python代码时可能没有遇到过的有趣的挑战。但不要担心，我们将尽量让问题简单。

1. 编写Cython代码。

我们将编写的Cython代码看上去和常规的Python代码差不多，只是把函数参数和一个局部变量声明为ndarray类型。为了使用静态类型，需要使用cimport引入NumPy，并且在声明局部变量时需要同时使用cdef关键字。

```
import numpy
cimport numpy

def ramanujan_factorial(numpy.ndarray n):
   sqrt_pi = numpy.sqrt(numpy.pi, dtype=numpy.float64)
   cdef numpy.ndarray root = (8 * n + 4) * n + 1
   root = root * n + 1/30.
   root = root ** (1/6.)
   return sqrt_pi * calc_eton(n) * root

def stirling_factorial(numpy.ndarray n):
    return numpy.sqrt(2 * numpy.pi * n) * calc_eton(n)

def calc_eton(numpy.ndarray n):
    return (n/numpy.e) ** n
```

2. 构建代码。

正如前面几个攻略中介绍过的，我们需要创建一个setup.py文件。在这个文件中，需要通过调用get_include函数，把NumPy相关的目录包括进来。修改之后的setup.py文件的内容如下。

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext
import numpy

ext_modules = [Extension("factorial", ["factorial.pyx"], include_dirs =
[numpy.get_include()])]

setup(
        name = 'Factorial app',
        cmdclass = {'build_ext': build_ext},
        ext_modules = ext_modules
     )
```

3. 绘制相对误差。

相对于用NumPy的cumprod函数计算出来的阶乘，我们将分别计算这两种近似计算方法的相对误差，并绘制出结果。

```
from factorial import ramanujan_factorial
from factorial import stirling_factorial
```

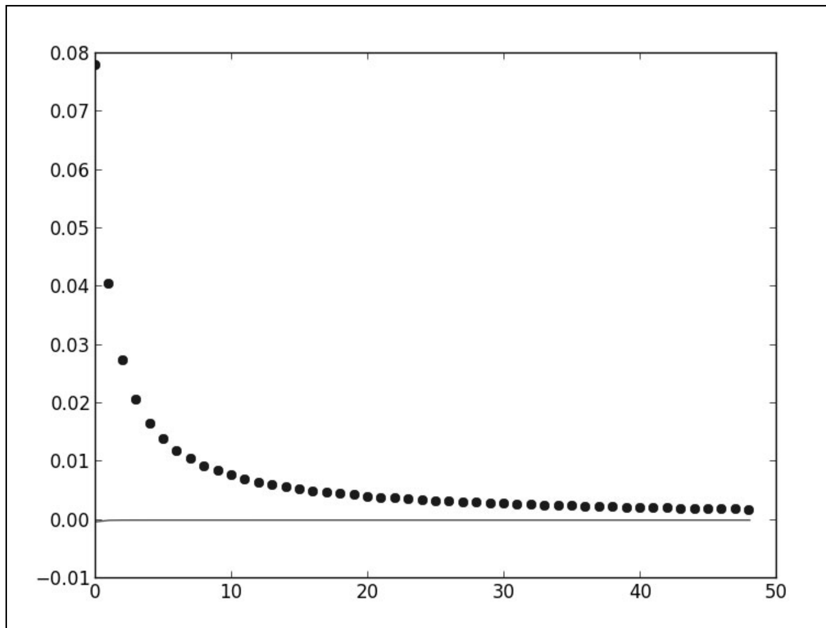
```
import numpy
import matplotlib.pyplot

N = 50
numbers = numpy.arange(1, N)
factorials = numpy.cumprod(numbers, dtype=float)

def error(approximations):
   return (factorials - approximations)/factorials

matplotlib.pyplot.plot(error(ramanujan_factorial(numbers)), 'b-')
matplotlib.pyplot.plot(error(stirling_factorial(numbers)), 'ro')
matplotlib.pyplot.show()
```

两种近似计算方法的相对误差图示如下。连续直线代表Ramanujan近似方法，点状线代表斯特灵近似方法。如你所见，Ramanuja近似方法比斯特灵近似方法更精确。



9.7.2　攻略小结

本攻略演示了怎样在Cython中使用静态类型。本攻略的要点如下。

- 用`cimport`引入C语言库
- 用`get_include`函数把指定目录包括进来
- 用`cdef`关键字修饰局部变量的类型

第 10 章

有趣的Scikits

本章主要内容：

- 安装scikits-learn
- 加载范例数据集
- 用scikits-learn对道琼斯成分股做聚类分析
- 安装scikits-statsmodels
- 用scikits-statsmodels做正态性检验
- 安装scikits-image
- 检测角点
- 检测边缘
- 安装pandas
- 用pandas估计股票收益的相关性
- 从statsmodels加载数据到`pandas`对象
- 重采样时间序列数据

10.1 引言

Scikits包含若干独立的小项目。它们和SciPy有一定联系，但不是SciPy的组成部分。这些项目之间并不完全独立，它们都以Scikits的名义运作，看上去有点像一个联盟。在本章中，我们将讨论下列几个Scikits项目：

- scikits-learn，机器学习包
- scikits-statsmodels，统计包
- scikits-image，图像处理包
- pandas，数据分析包

10.2　安装 scikits-learn

scikits-learn[①]项目提供了机器学习相关的API。这个项目的文档非常棒，这一点最让我喜欢。可以用操作系统自带的包管理器安装scikits-learn，这是最方便的安装途径。但这种安装方式是不是可用，要看你用的什么操作系统了。

Windows用户可以从scikits-learn项目的官网下载一个安装文件。在Debian和Ubuntu上，对应的包的名字是python-sklearn。在MacPorts上，对应的port的名字是py26-scikits-learn和py27-scikits-learn。我们也可以选择从源文件安装，或者使用easy_install安装。Python(x, y)、Enthought和NetBSD等第三方的发行版中也包含了scikits-learn。

10.2.1　准备工作

你需要提前安装好SciPy和NumPy。如有必要，请参阅第1章的相关内容。

10.2.2　具体步骤

让我们看看怎样安装scikits-learn。

▶ 使用easy_install或pip安装

在命令行界面中，键入如下两条命令之一完成安装。

```
pip install -U scikit-learn
easy_install -U scikit-learn
```

可能需要在上述命令之前添加sudo，或者以管理员身份登录，否则可能会因为权限不够，不能成功执行安装命令。

▶ 从源文件安装

从https://pypi.python.org/pypi/scikit-learn/下载源文件，解压缩后进入主目录，然后执行如下命令。

```
python setup.py install
```

① scikits-learn已经更名为scikit-learn。——译者注

10.3　加载范例数据集

scikits-learn项目中包含了若干数据集和范例图像，可以用来做一些实验。本攻略中，我们将加载一个scikits-learn发行版中的范例数据集。该数据集用二维的NumPy数组保存数据，其中还包括了与这些数据有关联的元数据。

具体步骤

我们将加载一个波士顿房价的范例数据集。这是一个很小的数据集，所以，如果你正在波士顿物色房屋，别太激动。有关范例数据集的更多的介绍请见http://scikit-learn.org/dev/modules/classes.html#module-sklearn.datasets。

我们将查看原始数据的形状，以及其中的最大值和最小值。形状对应一个元组，表示NumPy数组的维度信息。对目标数组，我们也做相同的操作。目标数组中包含了作为学习目标的数值。下面的代码实现了上述功能。

```
from sklearn import datasets

boston_prices = datasets.load_boston()
print "Data shape", boston_prices.data.shape
print "Data max=%s min=%s" %
  (boston_prices.data.max(),
  boston_prices.data.min())
print "Target shape",
  boston_prices.target.shape
print "Target max=%s min=%s" %
  (boston_prices.target.max(),
  boston_prices.target.min())
```

该程序的输出结果如下。

```
Data shape (506, 13)
Data max=711.0 min=0.0
Target shape (506,)
Target max=50.0 min=5.0
```

10.4　用 scikits-learn 对道琼斯成分股做聚类分析

聚类（clustering）代表一类机器学习算法，用来基于相似度对研究对象分组。本例将使用道琼斯工业指数成分股的对数收益率数据进行聚类分析。本攻略中的大多数步骤在之前的章节中已有介绍。

10.4.1　具体步骤

首先需要从雅虎财经频道下载这些股票的盘后数据。然后，计算平方亲和度矩阵（square affinity matrix）。最后，用`AffinityPropagation`类对股票进行聚类分析。

1. 下载股价数据。

使用道琼斯工业指数成分股代码下载2011年的股价数据。本例中，我们只对收盘价感兴趣。

```
# 2011到2012
start = datetime.datetime(2011, 01, 01)
end = datetime.datetime(2012, 01, 01)

#道琼斯工业指数成分股代码
symbols = ["AA", "AXP", "BA", "BAC", "CAT",
   "CSCO", "CVX", "DD", "DIS", "GE", "HD",
   "HPQ", "IBM", "INTC", "JNJ", "JPM", "KFT",
   "KO", "MCD", "MMM", "MRK", "MSFT", "PFE",
   "PG", "T", "TRV", "UTX", "VZ", "WMT", "XOM"]

quotes = [finance.quotes_historical_yahoo
  (symbol, start, end, asobject=True)
    for symbol in symbols]

close = numpy.array([q.close for q in quotes]).astype(numpy.float)
```

2. 计算亲和度矩阵。

把对数收益率作为度量值，计算不同股票之间的相似度。我们试图做的是计算数据点之间的欧式距离。

```
logreturns = numpy.diff(numpy.log(close))
print logreturns.shape

logreturns_norms = numpy.sum(logreturns ** 2, axis=1)
S = - logreturns_norms[:, numpy.newaxis] -
  logreturns_norms[numpy.newaxis, :] + 2 *
  numpy.dot(logreturns, logreturns.T)
```

3. 股票的聚类分析。

把上一步得到的结果提供给`AffinityPropagation`类。该类用来给数据点做标记，在本例中就是给股票标记适当的簇号。

```
aff_pro = sklearn.cluster.AffinityPropagation().fit(S)
labels = aff_pro.labels_

for i in xrange(len(labels)):
    print '%s in Cluster %d' % (symbols[i], labels[i])
```

完整的聚类程序如下。

```
import datetime
import numpy
import sklearn.cluster
from matplotlib import finance

#1. 下载股价数据

# 2011到2012
start = datetime.datetime(2011, 01, 01)
end = datetime.datetime(2012, 01, 01)

#道琼斯工业指数成分股代码
symbols = ["AA", "AXP", "BA", "BAC", "CAT",
   "CSCO", "CVX", "DD", "DIS", "GE", "HD",
   "HPQ", "IBM", "INTC", "JNJ", "JPM", "KFT",
   "KO", "MCD", "MMM", "MRK", "MSFT", "PFE",
   "PG", "T", "TRV", "UTX", "VZ", "WMT", "XOM"]

quotes = [finance.quotes_historical_yahoo(symbol, start, end, asobject=True)
          for symbol in symbols]

close = numpy.array([q.close for q in quotes]).astype(numpy.float)
print close.shape

#2. 计算亲和度矩阵
logreturns = numpy.diff(numpy.log(close))
print logreturns.shape

logreturns_norms = numpy.sum(logreturns ** 2, axis=1)
S = - logreturns_norms[:, numpy.newaxis] - logreturns_norms[numpy.newaxis, :] + 2 *
numpy.dot(logreturns, logreturns.T)

#3. 亲和传播聚类
aff_pro = sklearn.cluster.AffinityPropagation().fit(S)
labels = aff_pro.labels_

for i in xrange(len(labels)):
    print '%s in Cluster %d' % (symbols[i], labels[i])
```

程序的输出结果列出了股票代码及其对应的簇号，如下所示。

```
(30, 252)
(30, 251)
AA in Cluster 0
AXP in Cluster 6
BA in Cluster 6
BAC in Cluster 1
CAT in Cluster 6
CSCO in Cluster 2
CVX in Cluster 7
```

```
DD in Cluster 6
DIS in Cluster 6
GE in Cluster 6
HD in Cluster 5
HPQ in Cluster 3
IBM in Cluster 5
INTC in Cluster 6
JNJ in Cluster 5
JPM in Cluster 4
KFT in Cluster 5
KO in Cluster 5
MCD in Cluster 5
MMM in Cluster 6
MRK in Cluster 5
MSFT in Cluster 5
PFE in Cluster 7
PG in Cluster 5
T in Cluster 5
TRV in Cluster 5
UTX in Cluster 6
VZ in Cluster 5
WMT in Cluster 5
XOM in Cluster 7
```

10.4.2　攻略小结

下表的内容是对本攻略所用函数的概述。

函　　数	功能描述
`sklearn.cluster.AffinityPropagation()`	创建一个`AffinityPropagation`对象
`sklearn.cluster.AffinityPropagation.fit`	用欧式距离计算亲和度矩阵，并应用亲和传播聚类算法
`diff`	计算NumPy数组中数字间的差值。如果没有特别指明，默认计算一阶差分
`log`	计算NumPy数组中各个元素的自然对数
`sum`	对NumPy数组中的各个元素求和
`dot`	对二维数组，做矩阵乘法。对一维数组，则做内积运算

10.5　安装 scikits-statsmodels

scikits-statsmodels包聚焦于统计建模。它可以方便地与NumPy和Pandas集成（关于Pandas的更多内容将在后续攻略中介绍）。

具体步骤

可以从http://statsmodels.sourceforge.net/install.html下载scikits-statsmodels的源代码和二进制安装文件。如果选择从源文件安装，你需要运行如下命令。

```
python setup.py install
```

如果使用setuptools安装，相应的安装命令是：

```
easy_install statsmodels
```

10.6 用 scikits-statsmodels 做正态性检验

scikits-statsmodels包囊括了大量的统计检验方法。作为范例，我们将介绍正态性Anderson-Darling检验（http://en.wikipedia.org/wiki/Anderson%E2%80%93Darling_test）。

10.6.1 具体步骤

和上一篇攻略一样，我们将下载股价数据，但只需要下载一支股票的数据。然后我们将计算这支股票收盘价的对数收益率，并且把计算结果作为正态性检验函数的输入数据。

该函数返回一个元组，元组中的第二个元素是一个取值在0到1之间的p值（p-value）。

本攻略的完整代码如下。

```
import datetime
import numpy
from matplotlib import finance
from statsmodels.stats.adnorm import normal_ad
import sys

#1. 下载股价数据

# 2011到2012
start = datetime.datetime(2011, 01, 01)
end = datetime.datetime(2012, 01, 01)

print "Retrieving data for", sys.argv[1]
quotes = finance.quotes_historical_yahoo(sys.argv[1], start, end, asobject=True)

close = numpy.array(quotes.close).astype(numpy.float)
print close.shape

print normal_ad(numpy.diff(numpy.log(close)))
```

该脚本的输出结果如下所示。我们得到的p值是0.13。

```
Retrieving data for AAPL
(252,)
(0.57103805516803163, 0.13725944999430437)
```

10.6.2　攻略小结

本攻略示范了怎样用scikits-statsmodels中的Anderson-Darling统计检验方法做正态性检验。我们把不符合正态分布的股价数据作为Anderson-Darling检验函数的输入，得到的p值是0.13，这一结果也证实了我们的假设。

10.7　安装 scikits-image

scikits-image是一个图像处理工具包，需要用到PIL、SciPy、Cython和NumPy。该软件包有Windows版本的安装文件可用。在EPD（Enthought Python Distribution）和Python(x, y)发行版中已经包含了该软件包。

具体步骤

可以用以下两条命令之一完成安装。[①]

```
pip install -U scikits-image
easy_install -U scikits-image
```

你可能需要以root身份运行上述命令。

你也可以通过复制Git版本库的方式获得最新的开发版本，或者从Github以zip文件的方式下载版本库。之后，你需要运行如下命令。

```
python setup.py install
```

10.8　检测角点

角点检测（http://en.wikipedia.org/wiki/Corner_detection）是计算机视觉领域的一项基本技术。角点检测是相当复杂的，而scikit-image提供了Harris角点检测器，这真是太棒了。显然，我们也可以选择从头开始自己来实现角点检测算法，但那样将违背“不做无谓的重复劳动”（not reinventing the wheel）的基本准则。

① scikits-image已经更名为scikit-image。下载最新版本时需要用scikit-image。——译者注

10.8.1 准备工作

你可能需要先安装jpeglib，以便加载scikit-learn中JPEG格式的图像。在Windows系统中，可以使用安装文件安装jpeglib。否则需要下载jpeglib的发行版，解压缩后在主目录中用如下命令完成构建和安装。

```
./configure
  make
    sudo make install
```

10.8.2 具体步骤

我们将从scikit-learn加载一幅范例图像。这一步不是绝对必要的，你也可以使用任何其他的图像替代。

1. 加载范例图像。

scikit-learn当前包含了两幅JPEG格式的范例图像，存储在一个数据集结构中。我们只加载第一幅图像。

```
dataset = load_sample_images()
img = dataset.images[0]
```

2. 检测角点。

调用harris函数[①]，用来获取角点坐标。

```
harris_coords = harris(img)
print "Harris coords shape", harris_coords.shape
y, x = numpy.transpose(harris_coords)
```

角点检测的完整代码如下。

```
from sklearn.datasets import load_sample_images
from matplotlib.pyplot import imshow, show, axis, plot
import numpy
from skimage.feature import harris

dataset = load_sample_images()
img = dataset.images[0]
harris_coords = harris(img)
print "Harris coords shape", harris_coords.shape
y, x = numpy.transpose(harris_coords)
```

① 运行环境中的scikit-image版本号为0.6、0.7时，该程序运行正常。0.8及之后的版本去掉了harris函数，需要改用corner_harris函数和corner_peaks函数。——译者注

```
axis('off')
imshow(img)
plot(x, y, 'ro')
show()
```

在我们得到的图像中，检测到的角点被标记为红点，具体如下图所示。



10.8.3　攻略小结

我们把Harris角点检测算法应用于scikit-learn中的一幅范例图像。如你所见，检测结果是相当令人满意的。因为角点检测算法其实就是一个直截了当的线性代数类型的计算，所以我们也可以只用NumPy来做相同的事情，但代码看上去会非常混乱。scikit-image工具包中有很多类似的函数可用，所以当你需要编写一个图像处理程序时，最好先查看一下scikit-image的文档。

10.9　检测边缘

边缘检测（http://en.wikipedia.org/wiki/Edge_detection）是另一项流行的图像处理技术。scikit-image实现了一个Canny滤波器。该滤波器基于高斯分布的标准差，可以直接用来做边缘检测。除了二维数组中的图像数据，该滤波器还接受下列参数：

- 高斯分布的标准差
- 下边界阈值
- 上边界阈值

具体步骤

我们使用的图像和上一攻略相同，代码实现也基本相同。你只需要特别注意调用Canny滤波器函数的那一行代码。

```
from sklearn.datasets import load_sample_images
from matplotlib.pyplot import imshow, show, axis
import numpy
import skimage.filter

dataset = load_sample_images()
img = dataset.images[0]
edges = skimage.filter.canny(img[..., 0], 2, 0.3, 0.2)
axis('off')
imshow(edges)
show()
```

上述代码生成了一幅由原始图像中的边缘构成的图像，具体如下图所示。



10.10 安装 Pandas

Pandas是Python的一个数据分析库。Pandas和R编程语言有几分相似，这不是巧合。R是为广大数据分析专家所喜爱的专用编程语言。Pandas中核心的`DataFrame`对象，其设计灵感就来源于R。

具体步骤

Pandas项目在PyPi上对应的包名就是pandas。运行以下两条命令之一完成安装。

```
sudo easy_install -U pandas
pip install pandas
```

如果你在使用Linux的包管理器，需要安装python-pandas项目。例如，在Ubuntu中，需要运行如下命令。

```
sudo apt-get install python-pandas
```

你也可以选择从源文件安装，这需要用到Git。

```
git clone git://github.com/pydata/pandas.git
cd pandas
python setup.py install
```

10.11　用 Pandas 估计股票收益的相关性

Pandas中的DataFrame是一个兼有矩阵和字典特征的数据结构，其功能类似于R中的dataframe。实际上，DataFrame是Pandas中的核心数据结构，你可以对其做各种操作。进行投资组合分析时，经常需要计算投资组合的相关矩阵，本攻略将对此予以介绍。

10.11.1　具体步骤

首先，创建一个字典对象，其成员是每支股票的日收益率数据。然后，以日期为行标签，创建DataFrame对象。最后，计算相关矩阵并绘制结果。

1. 创建DataFrame对象。

为了创建DataFrame对象，需要先创建一个字典。该字典对象以股票代码为索引关键字（键），对应的值是该股的对数收益率。该DataFrame对象以日期为索引（行标签），以股票代码为列标签。

```
data = {}

for i in xrange(len(symbols)):
  data[symbols[i]] = numpy.diff(numpy.log(close[i]))

df = pandas.DataFrame(data,
  index=dates[0][:-1], columns=symbols)
```

2. 操作DataFrame对象。

现在可以对DataFrame对象做各种操作，例如计算相关矩阵或绘图。

```
print df.corr()
df.plot()
```

包括下载股价数据在内的完整代码如下。

```
import pandas
from matplotlib.pyplot import show, legend
from datetime import datetime
from matplotlib import finance
import numpy

# 2011到2012
start = datetime(2011, 01, 01)
end = datetime(2012, 01, 01)

symbols = ["AA", "AXP", "BA", "BAC", "CAT"]

quotes = [finance.quotes_historical_yahoo(symbol, start, end, asobject=True)
          for symbol in symbols]

close = numpy.array([q.close for q in quotes]).astype(numpy.float)
dates = numpy.array([q.date for q in quotes])

data = {}

for i in xrange(len(symbols)):
   data[symbols[i]] = numpy.diff(numpy.log(close[i]))

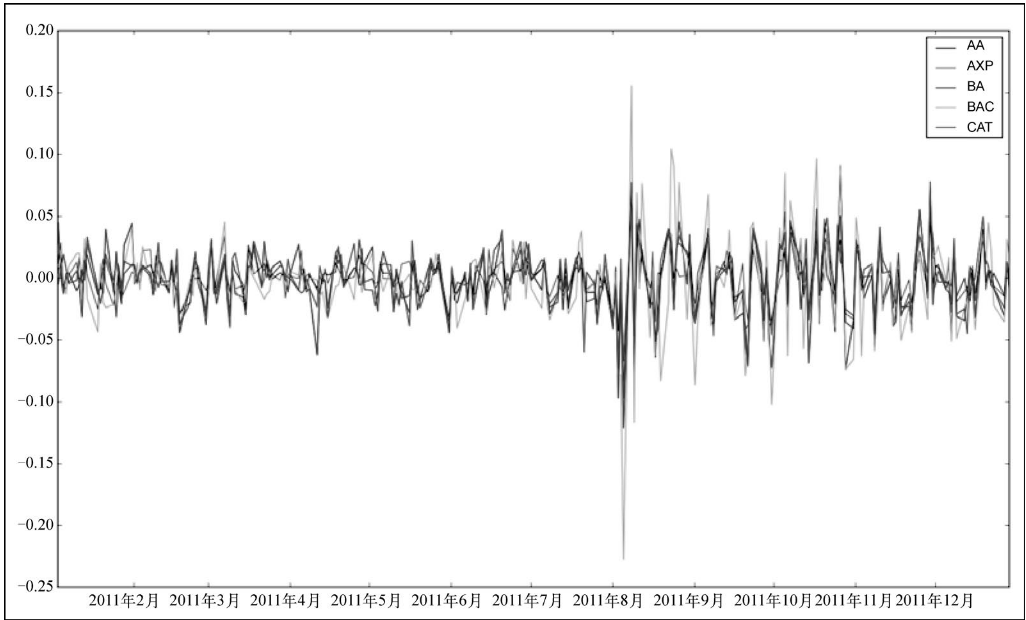
df = pandas.DataFrame(data, index=dates[0][:-1], columns=symbols)


print df.corr()
df.plot()
legend(symbols)
show()
```

相关矩阵的输出结果如下。

```
          AA       AXP        BA       BAC       CAT
AA   1.000000  0.768484  0.758264  0.737625  0.837643
AXP  0.768484  1.000000  0.746898  0.760043  0.736337
BA   0.758264  0.746898  1.000000  0.657075  0.770696
BAC  0.737625  0.760043  0.657075  1.000000  0.657113
CAT  0.837643  0.736337  0.770696  0.657113  1.000000
```

上述五支股票的对数收益率图示如下。



10.11.2　攻略小结

我们用到了`DataFrame`对象的下列方法。

方法名	功能描述
`pandas.DataFrame`	用指定的数据、索引（行标签）和列标签构造`DataFrame`对象
`pandas.DataFrame.corr`	各个列两两之间计算相关系数，计算过程忽略缺失值。默认使用Pearson相关系数
`pandas.DataFrame.plot`	用Matplotlib绘制`DataFrame`对象

10.12　从 statsmodels 加载数据到 pandas 对象

Statsmodels的发行版中有很多的范例数据集。可以在https://github.com/statsmodels/statsmodels/tree/master/statsmodels/datasets找到这些数据集的完整列表。

在本攻略中，我们将聚焦copper数据集，其中包含的信息有：铜价、全球铜消费量和其他参数。

10.12.1　准备工作

你可能需要先安装patsy。判断是否需要安装patsy很容易，只需运行一下本攻略的完整代码。如果得到的运行结果是和pasty有关的错误信息，则需要执行如下两条命令之一。

```
sudo easy_install patsy
pip install --upgrade patsy
```

10.12.2　具体步骤

在本节中，你将看到怎样从statsmodels加载数据集到Pandas的DataFrame对象或Series对象。

1. 加载数据。

需要调用的函数是load_pandas。用如下语句加载数据。

```
data = statsmodels.api.datasets.copper.load_pandas()
```

这将生成一个DataSet对象，数据被加载到了该对象包含的pandas对象中。

2. 拟合数据。

对于数据以pandas对象加载的情况，使用Dataset对象的exog属性，可以获得一个多列的DataFrame对象。此外对于本例的情况，DataSet对象的endog属性包含了全球铜消费量的数据。

采用最小二乘拟合，创建一个OLS（ordinary least squares）对象，并调用其fit方法。

```
x, y = data.exog, data.endog

fit = statsmodels.api.OLS(y, x).fit()
print "Fit params", fit.params
```

打印出拟合的参数，如下所示。

```
Fit params COPPERPRICE          14.222028
INCOMEINDEX        1693.166242
ALUMPRICE           -60.638117
INVENTORYINDEX     2515.374903
TIME                183.193035
```

3. 汇总结果。

使用summary方法，对最小二乘拟合的结果进行汇总，如下所示。

```
print fit.summary()
```

打印出回归分析的结果。

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       WORLDCONSUMPTION   R-squared:                       0.978
Model:                            OLS   Adj. R-squared:                  0.974
Method:                 Least Squares   F-statistic:                     224.9
Date:                Fri, 28 Sep 2012   Prob (F-statistic):           2.55e-16
Time:                        20:25:26   Log-Likelihood:                -172.62
No. Observations:                  25   AIC:                             355.2
Df Residuals:                      20   BIC:                             361.3
Df Model:                           4
==================================================================================
                     coef    std err          t      P>|t|      [95.0% Conf. Int.]
----------------------------------------------------------------------------------
COPPERPRICE       14.2220     12.090      1.176      0.253      -10.998     39.442
INCOMEINDEX     1693.1662   1970.555      0.859      0.400    -2417.339   5803.671
ALUMPRICE        -60.6381     32.023     -1.894      0.073     -127.437      6.161
INVENTORYINDEX  2515.3749   1670.948      1.505      0.148     -970.162   6000.912
TIME             183.1930     36.879      4.967      0.000      106.264    260.122
==============================================================================
Omnibus:                        8.007   Durbin-Watson:                   1.316
Prob(Omnibus):                  0.018   Jarque-Bera (JB):                6.014
Skew:                          -0.936   Prob(JB):                       0.0494
Kurtosis:                       4.506   Cond. No.                     2.20e+03
==============================================================================

The condition number is large, 2.2e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

加载copper数据集的完整代码如下。

```
import statsmodels.api

# 请见https://github.com/statsmodels/statsmodels/tree/master/statsmodels/datasets
data = statsmodels.api.datasets.copper.load_pandas()

x, y = data.exog, data.endog

fit = statsmodels.api.OLS(y, x).fit()
print "Fit params", fit.params
print
print "Summary"
print
print fit.summary()
```

10.12.3　攻略小结

statsmodels中的`Dataset`类采用了特殊的数据存储格式，这其中就包括了`endog`和`exog`属性。statsmodels中有一个`load`函数，用来把数据加载到NumPy数组中。而本攻略选用的`load_pandas`方法，则是用来把数据加载到`pandas`对象中。我们对这些数据做了最小二乘拟合，得到了铜价和消费量之间关系的统计模型。

10.13 重采样时间序列数据

在本攻略中，我们将学习怎样使用Pandas重采样时间序列数据。

10.13.1 具体步骤

我们将下载AAPL股票每日股价的时间序列数据，然后通过计算平均值的方式，对这些数据做重采样处理。为此，我们将创建一个Pandas的DataFrame对象，并调用它的resample方法。

1. 创建一个日期时间索引对象。

在创建Pandas的DataFrame对象之前，需要先创建一个日期时间索引（DatetimeIndex）对象。这个DatetimeIndex对象将被作为索引传递给DataFrame的构造函数。用如下语句从下载的股价数据创建日期时间索引对象。

```
dt_idx = pandas.DatetimeIndex(quotes.date)
```

2. 创建DataFrame对象。

用日期时间索引对象和收盘价数据创建一个DataFrame对象。

```
df = pandas.DataFrame(quotes.close, index=dt_idx,
  columns=[symbol])
```

3. 重采样。

以每月一次的频率，通过计算平均值的方式，重采样时间序列数据。

```
resampled = df.resample('M', how=numpy.mean)
print resampled
```

重采样后的时间序列中，每月只有一个数据，如下所示。

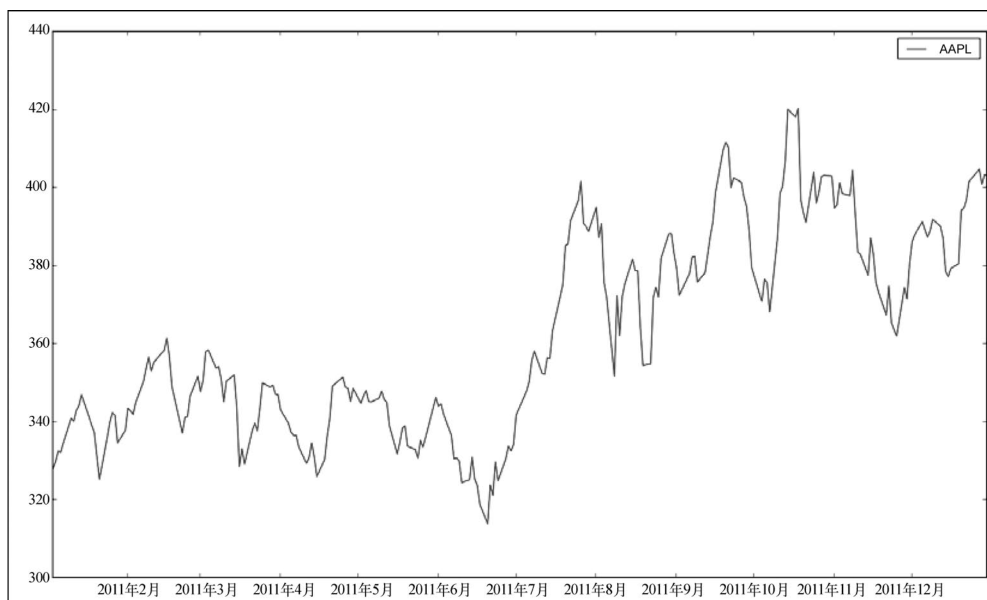
```
                  AAPL
2011-01-31 336.932500
2011-02-28 349.680526
2011-03-31 346.005652
2011-04-30 338.960000
2011-05-31 340.324286
2011-06-30 329.664545
2011-07-31 370.647000
2011-08-31 375.151304
2011-09-30 390.816190
2011-10-31 395.532381
2011-11-30 383.170476
2011-12-31 391.251429
```

4. 绘图。

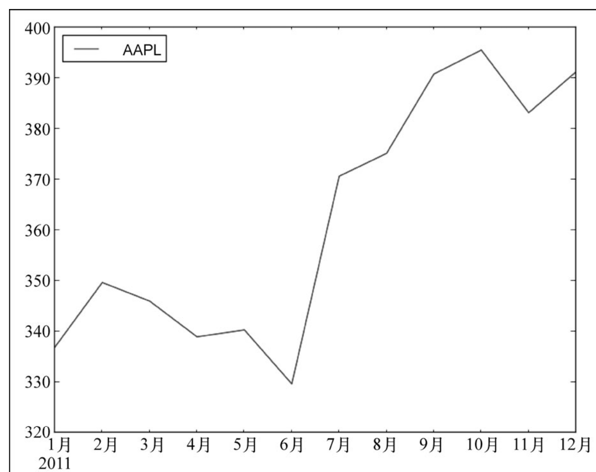
用DataFrame的plot方法，绘制数据。

```
df.plot()
resampled.plot()
show()
```

用原始时间序列绘图后的结果如下图所示。



重采样后的时间序列数据点较少，因此用该序列绘制的曲线看上去不太连贯，如下图所示。



完整的重采样代码如下。

```
import pandas
from matplotlib.pyplot import show, legend
from datetime import datetime
from matplotlib import finance
import numpy

# 下载2011年到2012年的AAPL股价数据
start = datetime(2011, 01, 01)
end = datetime(2012, 01, 01)

symbol = "AAPL"
quotes = finance.quotes_historical_yahoo(symbol, start, end, asobject=True)

# 创建日期时间索引对象
dt_idx = pandas.DatetimeIndex(quotes.date)

# 创建DataFrame对象
df = pandas.DataFrame(quotes.close, index=dt_idx, columns=[symbol])

# 以每月一次的频率重采样
resampled = df.resample('M', how=numpy.mean)
print resampled

# 绘图
df.plot()
resampled.plot()
show()
```

10.13.2 攻略小结

我们用日期和时间列表，创建了一个日期时间索引对象。该索引对象被用来创建Pandas的DataFrame对象。我们对时间序列数据进行了重采样，重采样的频率通过一个单字符的参数给出：

- D，每天一次
- M，每月一次
- A，每年一次

resample方法中的how参数指明了具体的重采样方式，默认的重采样方式是计算平均值。

索　　引

A

B

C

D

E

F

G

H

I

J

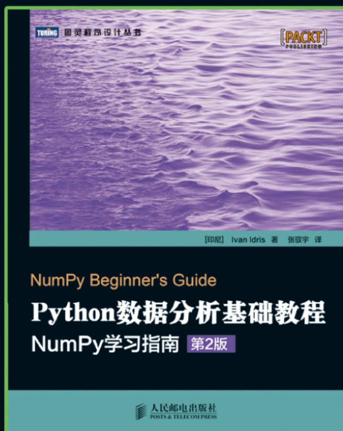
K

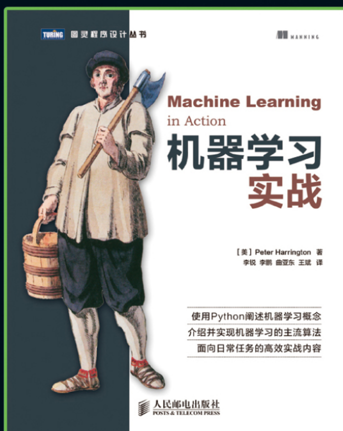
L

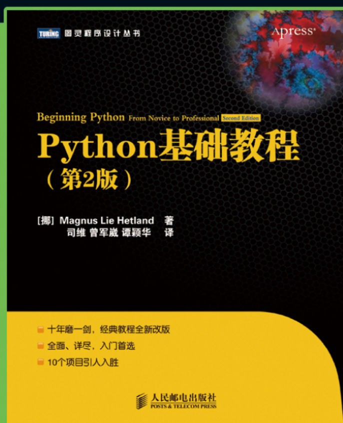
M

N

O


图灵程序设计丛书
[PACKT]
[印度] Ivan Idris 著 张驭宇 译
NumPy Beginner's Guide
Python数据分析基础教程
NumPy学习指南 第2版
人民邮电出版社
POSTS & TELECOM PRESS


图灵程序设计丛书
Machine Learning
in Action
机器学习
实战
[美] Peter Harrington 著
使用Python阐述机器学习概念
介绍并实现机器学习的主流算法
面向日常任务的高效实战内容
人民邮电出版社
POSTS & TELECOM PRESS


图灵程序设计丛书
Apress
Beginning Python From Novice to Professional Second Edition
Python基础教程
（第2版）
[挪] Magnus Lie Hetland 著
司维 曾军崴 谭颖华 译
十年磨一剑，经典教程全新改版
全面、详尽，入门首选
10个项目引人入胜
人民邮电出版社
POSTS & TELECOM PRESS

“本书写得非常棒，它能告诉你使用NumPy时的常见问题，并列出了简明的解决方案。这些方案并非仅针对你眼下遇到的难题，而是为你打开了一扇门，让你能够洞察NumPy世界的深邃迷人之处。”

“我刚刚开始学习使用Python、NumPy、SciPy和Matplotlib，本书给了我巨大的帮助！我觉得书中的示例非常详尽，并且覆盖了非常宽的领域。这些领域也许你不会全部感兴趣，但其中用到的技术可以轻松解决你目前遇到的各种问题，为你节省大量时间。”

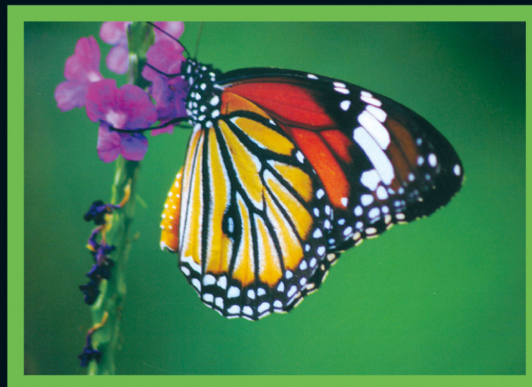
“书中的攻略详尽而且具体，对Python和NumPy的学习大有裨益。其示例丰富至极，能够帮助你拓展思维，让你明白NumPy的适用之处，这点非常令人振奋！未来我会一直使用这本极佳的参考书！”

——亚马逊读者评论

本书带领读者了解熟悉当下最流行的科学计算库NumPy的方方面面。书中不仅介绍了NumPy的安装、使用和各种相关概念，还介绍了如何利用这一最新的开源软件库，以尽可能接近传统数学语言的方式，编写可读性好、实现效率高和运行速度快的代码。最后还探究了几个和NumPy相关的科学计算项目。此外，本书将为你掌握NumPy数组和通用函数打下坚实的基础，也会通过实例教你用Matplotlib绘图，并了解和SciPy相关的项目。

本书将帮助你：

- 学习高级索引技术和线性代数知识
- 了解数组形状及图像大小的调整
- 探察广播机制和直方图
- 分析NumPy代码并用可视化的方式表示分析结果
- 用Cython为代码提速
- 使用数组接口共享数据
- 使用通用函数和互操作功能
- 学习Matplotlib以及经常和NumPy同时使用的SciPy





[PACKT] PUBLISHING

图灵社区：www.ituring.com.cn
新浪微博：@图灵教育 @图灵社区
反馈/投稿/推荐信箱：contact@turingbook.com
热线：(010)51095186转604
分类建议 计算机/软件开发
人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-32991-2
定价：49.00元